Using FTP, Views, and PROC SUMMARY to analyse large databases.

Don Stanley
Don Stanley Consulting Limited
Wellington
NEW ZEALAND
EMAIL: Don_Stanley@ibm.net

Overview

This paper discusses some techniques that have proven helpful to the New Zealand Accident Rehabilitation and Compensation Corporations (ACC) recent analysis of data quality.

The ACC recently undertook a project to redevelop its core computer systems. Over the last few years, many different systems that provide ACC functionality have been reviewed, with the result that a team, called Data Transition, was formed in early 1996 to carry out the task of analyzing existing data, comparing data where duplicate data was stored in several systems, modeling existing databases and developing models for an optimal database. Following the analysis, the team is also charged with the tasks of data cleaning and developing extracts from existing systems to populate new databases. I joined the data transition team at end of MAY 96 after it had been in existence for 3 months.

The existing ACC systems are Fujitsu AIM databases. There are three main systems which often store similar data. For historical reasons, these 3 systems duplicate data, and have separate mechanisms for entering data. Interface programs run daily to keep the data synchronized, i.e. to move data from one system to the others.

In addition to the AIM databases, ACC runs a SUN 4 UNIX box primarily running SAS as a data warehouse. SAS does not run on the Fujitsu, so all analysis is done on the SUN, including that discussed here.

A feature of the AIM databases is their size. Some, such as the master person record, have about 3.6 million records. Much of this is duplicated in another ACC system. This is about a 700 megabyte SAS file on the SUN. Others, such as the database that describes all medical services by person, exceed 32 million records. Another, describing medical procedures, exceeds 96 million records. These latter two files are 3.5 gigs and 7.7 gigs respectively and do not form part of the SUN data warehouse. However, they are required for data transition analysis.

Data Analysis

Data transition have been charged with the task of analyzing AIM data to look at validity, completeness and correctness. A example here is to verify on the person database that the sex is consistent with the persons title. Another example is that the stored calculated SOUNDX field used for person searching actually is the correct soundx.

Many of the things data transition are looking at have come from knowledge of the ACC system within ACC operations staff, some have been inspired by data transition when looking at other things (for example we were looking at birth date versus claim date to ensure claims were entered after birth, and found a number of people born prior to 1850 which spawned a separate analysis). The business have also requested certain analysis, the soundx on surname mentioned above was investigated because of a perception in the business branches that searches didn't work properly.

This paper discusses a specific analysis that data transition had to do. This involved the 32 million plus record medical services file. To carry out this analysis I had to go back to basics and examine the whole way that data transition had traditionally been working.

SAS At ACC

From a purely data transition team viewpoint, the use of SAS on the SUN analyzing Fujitsu data was somewhat inefficient.

Warehouse and MIS data were summarized too much for data transition to make any use of it. Also, warehouse data only cover the latest financial year, and our exercise involved the AIM databases back to ACC's inception in 1974.

To do an analysis, data transition would write an INTERACT query on the Fujitsu. INTERACT is an AIM utility that writes AIM database fields to a flat file. It follows the AIM field storage structures exactly, i.e. a signed packed field in AIM is written out signed packed to the flat file by INTERACT.

The INTERACT queries are simple to create, but often take 2-3 hours to run and are a cpu hog. They are unavoidable as they are the only means of getting data from AIM to a flat file.

Following the INTERACT query, data transition were using FTP to transfer the flat file, in its entirety, from Fujitsu to SUN. This could take anything from 2-5 hours dependent on SUN and Fujitsu loading. It is very CPU intensive. It is also very inefficient on SUN disk.

The inefficiency inherent in this on SUN disk space is that at least two copies must be stored on the SUN. This is the downloaded flat file, plus the work space that SAS requires. Additionally, we usually wanted to keep the file after reading on the SUN. The chain of events

that occurred was very inefficient and followed these steps:

- Interact — 2-4 hours to get an up to date 3.5 million record file
- FTP — 2 hours to transmit to SUN
- SAS — 1-2 hours to read on SUN and create SAS file

Then we could start analyzing the file. Note that these times were not absolute, they depended on system loading and we often struck a situation where the Fujitsu ground to a halt when downloading using FTP. So sometimes the actual time to get data was measured in days, but the physical process of downloading when able to followed the above table.

This was for one of the simplest, smallest files. As file size increased the inherent deadtime in the FTP transfer meant that analyses were taking longer to perform.

Efficient Use of FTP

FTP (File Transfer Protocol) is a method of transmitting files across computers connected using a TCP/IP protocol. If you have multiple machines at your site you should contact your system administrator to see if they are connected by FTP.

In release 6.11, SAS Institute provide production support for treating a file on an external machine as an input file without first copying that file to the machine that SAS is running on. This is just another example of the client-server facilities that SAS provide. The machine running SAS is a client, and requests data from the server machine by means of an FTP request.

One of the most critical problems facing data transition was the deadtime inherent in moving a whole file via FTP from Fujitsu to SUN. So, by using SAS to read the file using FTP, the entire transfer time is removed from the equation. SAS now reads the file, then immediately does the analysis. In fact, because the transmit now sent each record direct to SAS, new fields could be created and various record counts started as the data was sent from the Fujitsu.

As well as eliminating the deadtime associated with the sending of the whole file then starting a SAS analysis, using FTP direct into SAS had another tremendous advantage. The SUN is disk constrained. We had a total of 4 gigs for all our files, including work space. Four data transition team members regularly fought for space. By removing the copy of the flat file (actually many flat files as we usually each worked on different extracts), we were able to vastly improve the disk utilization.

Having FTP direct into SAS had other spin-offs. Some people would regard these spin-offs as additional problems. The main one was that now we could look at analyzing files that previously could not be downloaded due to their size.

Incidentally, transmitting a file in its entirety is quite CPU intensive as the Fujitsu repeatedly hands the SUN blocks of data. Using SAS reduces this to some extent, the SUN is still sent data in a similar way, but now it also processes that data instead of just receiving it and writing it. This appears to cause a little less bottleneck on the Fujitsu, as the SUN takes longer between requests.

Handling Very Large Files

We soon found that our new found space was not enough. It allowed us to analyse bigger files, and more quickly, but ultimately just led back to the same situation as workspace requirements for even bigger files became difficult to cope with. So I started looking for different ways to achieve some of the analyses.

Consider the following analysis that we needed to carry out.

We have a master file containing about 8800 records. This is a list of people or companies who provide some service which ACC pays for (E.g. a doctor treating a patient whose injuries are eligible for ACC compensation). We wanted to find out the following from the 32 million record services database

- how many services have no provider in the master file
- how many providers have no services
- how many services were provided in each year from 1974 (a year is defined as 01JUL to 30JUN)
- percentiles for number of services per provider
- how many providers have most recent service prior to 1994

None of these questions is particularly difficult to answer. The processing involved to get the answers is basic and typical SAS number crunching. But the practical issues associated with space make the questions impossible to resolve using traditional techniques which may include downloading input files, creating SAS datasets and then analyzing.

SAS Views

A view is a template. Any datastep that normally produces a single dataset could instead be stored as a view. When SAS attempts to use the view (E.g. in a PROC PRINT), the datastep executes and instead of writing the records it creates to a SAS dataset, it instead passes them to the process that invoked the view.

One of the ramifications of this is that SAS can read a flat file and pass each record to a procedure without creating an intermediate dataset. The procedure can then

do its thing with the records. This implies that a substantial disk saving may be possible as no dataset ever needs to exist, and only scratch files created by the calling process will be used. At worst, you are likely to need half the space that would have been required with a dataset (and then only if the calling process needs to make a copy of the dataset).

Now, we already saw that SAS can read a file on the Fujitsu using FTP. If we use a view, the 32 million record file can be passed record by record to a procedure for analysis. Further, since the view is itself inherently a SAS datastep, we can create any extra fields that the procedure requires.

Essentially the problem here comes down to needing to summarize the records in the 32 million record database down to one record per service provider. By passing each record from the view straight to PROC SUMMARY, we should be able to almost completely remove the need for work space. This is because SUMMARY uses memory to build its classification levels.

When PROC SUMMARY runs against a view, the datastep inherent in the view effectively runs and PROC SUMMARY waits for it to return data. Then summary operates on that record as it normally would had the record come from a dataset.

Because the view just executes a datastep for a record, you can do anything in the view that can be done in a standard datastep. This includes creating new fields.

The ability to create new fields in the view is absolutely critical to the reduction of work space. Had we not been able to do that, we would have achieved nothing, a copy of the flat file in SAS dataset format would have been needed and we would have simply not had the space.

Remember one of the requirements here is to find out many service records existed per year. To do this, I created, in the view, a new variable for each year. This is easiest achieved as follows:

```
if '0JUL96'd le servday le '30JUN96'd then
year96=1 ;
```

and creating a year field for each year of interest. We were interested here back to July 1990, with a catch all year of year89 being all data entered prior to that date. So I created a yearxx variable for each of these. These year fields can be passed into PROC SUMMARYS var list and used to count each years services.

In addition to getting the count per year, which is easily obtained by summing the YEARxx fields created in the view, we also require a number of extra statistics. These statistics include

- percentiles
- number of providers with no services since 1994
- services that have no provider (should be impossible, but data transition team have found that with the number of incarnations of these systems impossible is not a viable assumption)
- providers who never had a service

It is not obvious at first, but all these are obtainable from the dataset created by PROC SUMMARY. If we use the provider identifier as the class variable, then PROC SUMMARY will produce two distinctly different types of output records.

The first is the overall summary level, which will contain all the counts across the whole file, i.e. the statistics for the number of services per year. This is just one record. It is identified by the SAS internal field _TYPE_ being equal to 0 and also by being the first record. It will also be the only record with a blank classification variable.

The second type of record is a set of statistics grouped by provider id. There is one of these records for each provider on the 32 million record file. These are identified by the SAS internal field _TYPE_ being equal to 1.

Extracting The Statistics

At this stage I am going to start looking at code. First, issue a filename to establish a connection to your FTP file if necessary. Note that when connecting to a Fujitsu, for reasons that no-one has yet been able to answer, the filename statement needs to include a file structured as follows:

```
filename myfile ftp '/ziisds/qmfs.servday'
host=<hostname>  user=<username> prompt
lrecl=<record length>
blocksize=<blocksize> recfm=f ;
```

The actual filename is an unusual structure, but the more traditional MVS structure of 'ziisds.qmfs.servday' just doesn't seem to work. The lrecl, blocksize and recfm turned out critical; omit any and the datastep just doesn't work.

The following datastep view is used to read the FTP file and create relevant extra fields.

```
data bigfile / view=bigfile ;
 infile bigfile ;
 input  person $ebcdic8.
        injid  s370fpd2.
        medfnd $ebcdicl0.
        provider $ebcdic6.
        provclss $ebcdic2.
        servday s370fpd3.
        servid  s370fpd5.
        statis $ebcdicl. ;

 /* header record removal */

 if _n_ eq 1 then delete ;
```

```
 format servday ddmmyyl0. ;

 /* set condition fields */

 length  nservice sum96 sum95 sum94 sum93
         sum92 sum91 sum90 sumpre90 deleted
         3 ;

 if statis ne '*' then nservice = 1 ;

/* please note -- normally the following
if statements would be joined together
with an else. It is omitted here solely to
permit the code to fit in the column space
available */

 if statis ne '*' & 'Oljul95'd le servday
       le '30jun96'd then year96 = 1 ;
 if statis ne '*' & 'Oljul94'd le servday
       le '30jun95'd then year95 = 1 ;
 if statis ne '*' & 'Oljul93'd le servday
       le '30jun94'd then year94 = 1 ;
 if statis ne '*' & 'Oljul92'd le servday
       le '30jun93'd then year93 = 1 ;
 if statis ne '*' & 'Oljul9l'd le servday
       le '30jun92'd then year92 = 1 ;
 if statis ne '*' & 'Oljul90'd le servday
       le '30jun91'd then year91 = 1 ;
 if statis ne '*' & 'Oljul89'd le servday
       le '30jun90'd then year90 = 1 ;
 if statis ne '*' & servday lt 'Oljul89'd
       then year89 = 1 ;

/* end of code that should have else
statements */

 if statis eq '*' then deleted = 1 ;
run ;
```

In the above step, the statis field value of '*' indicates that a record is logically deleted. The count of logically deleted records is of interest because they effectively waste space in the AIM dataset.

The view creates the YEARxx fields, assigning a value of 1 or missing, plus the deleted field to count the logically deleted records, and overall count field (nservice) which is like a dummy field. It will have a value of 1 for every record. These fields will simply be summed in PROC SUMMARY to give the overall and provider group counts.

```
data provptyp ;
 infile '/home/datatran/prv96may.mfs'
       recfm=f lrecl=91 ;
 input @65 provider $ebcdic6. ;
 if  _n_ eq 1 then delete ;
run ;
```

This step just reads the master list of providers which is a small flat file (about 8800 records) already on the SUN. Only 1 field, the provider id is read. In the 32 million record file, the provider id is the field that is used in the PROC SUMMARY class statement, so this master file will be used later in the job to compare with the SUMMARY output _type_ =1 records to determine how many providers have no service, and how many services have no provider.

```
proc summary data=bigfile ;
  class provider ;
  var nservice year96 year95 year94 year93
      year92 year91 year90 year89
      deleted servday ;
  output out=provs(drop=_type_
       rename=(_freq_=totserv)) sum=
        max(servday) =mservday  ;
run ;
```

Note how the dataset name on the PROC SUMMARY is BIGFILE, which is the view created above. Remember that a view executes the when requested by a calling process, so this code is going to see PROC SUMMARY request a record from the view, the datastep inherent in they view execute and get data from the Fujitsu, and then hand that record back to the procedure, and then SUMMARY will incorporate the record in its summarization.

Most of the fields are simply summed. Since they just have values of 1 or missing summing the field values amounts to counting how many records matched the particular attribute. Additionally, the maximum value of the servday field (the service date) is extracted for each provider, this will be used shortly to get the number of providers who have had no services since 1994.

This PROC SUMMARY is incredibly efficient on our scarce disk resource. It used a total of just over half a megabyte of work space. Given we were concerned that we would hit problems with our 32 million records this is an impressive use of disk resource, clearly showing the benefit of using the view. PROC SUMMARY used 1 hour 5 seconds of CPU.

Many of the statistics required are found on the _type_=0 record from the SUMMARY. To print these statistics is quite simple, you could do it in a variety of ways. This example uses SQL, but could as easy been a data step or other procedure.

```
proc sql inobs=1 ;
   select year96 'Total 1995/96 Year',
          year95 'Total 1994/95 Year',
          year94 'Total 1993/94 Year',
          year93 'Total 1992/93 Year',
          year92 'Total 1991/92 Year',
          year91 'Total 1990/91 Year',
          year90 'Total 1989/90 Year',
          year89 'Total pre OlJUL89',
        deleted 'Total Logical Deletions',
        totserv 'Total Services Provided'
   from provs ;
quit ;
```

In this SQL the inobs option forces SQL to only read the first record. The statistics that were required relating to overall counts are all on that record, so the above SQL provides all the annual counts, plus deleted and total records.

To obtain the number of providers with last service over 2 years ago (i.e. before 01jul1994) is quite simple. Just extract from the summary provider records (which differ

from the overall record in that they have a provider id, the overall count doesn't) a count of those records where the maximum service day is prior to that date.

```
proc sql ;
  select count(*) 'Total providers with
       last service before 01jul94'
  from provs
  where mservday between 0 and '30jun94'd
and provider ne ' ' ;
```

That's as simple as you could get, and shows how creating the maximum statistic by provider in PROC SUMMARY has permitted the SUMMARY to give us multiple answers, namely the overall counts, and now the count of outdated providers.

We are interested in two important statistics relating to the integrity of the services data These are to find out if any services have no provider in the 8800 record provider master file, and whether any provider has never provided a service. Use the following SQL.

```
  select count(*)
       'Providers With No Service Data'
   from
   (select provider from provptyp
     EXCEPT
   select provider from provs
       (firstobs=2));

   select  count(*)
       'Services With No Provider Account'
       from
    (select provider from provs
       (firstobs=2)
      EXCEPT
    select provider from provptyp) ;
```

This has now provided the bulk of the statistics we required. Note the firstobs=2 in the SQL, this causes record 1 (the overall count record) to be bypassed. Thus these analyses work on the records that contain counts for each provider.

The final statistic that was required is the percentiles of number of services. Each provider has a field called nservice that contains the total services for that provider. Use proc univariate to extract the percentiles.

```
proc univariate data=provs(firstobs=2) ;
 var nservice ;
 id provider     ;
run ;
```

It is of interest to compare SQL with the SUMMARY procedure. There is virtually nothing that SUMMARY does that SQL cannot. Using the following SQL

```
proc sql;
  create table provs as
    select sum(deleted) as deleted,
           sum(nservice) as nservice,
           sum(year96) as year96,
           sum(year95) as year95,
           sum(year94) as year94,
           sum(year93) as year93,
           sum(year92) as year92,
           sum(year91) as year91,
           sum(year90) as year90,
           sum(year89) as year89,
           max(servday) as mservday,
           count(*) as totrecs
   from bigfile
   group by provider ;
quit;
```

we saw in excess of 600 megabytes of work space used, and 2 hours 35 seconds CPU. This is despite the same view being used. SQL processes data in a very different manner to SUMMARY, and I would hazard a guess that for straight forward SUMMARY like tasks SQL will always prove less efficient. That isn't to say you should never use SQL, the product allows many things to be done simply and elegantly that are otherwise difficult or impossible using other SAS routines.

A Few Extras

A few additional efficiency tips are provided here with examples.

- We required to merge an 8 million record database with an 800,000 record database. The merge was by a key field, EVENTNUM. Only those event numbers on the smaller database are to be kept. There are multiple records for each EVENTNUM on each database, and the small database contains about 8% of the keys in the big database.

  SQL joins were of little use, as they consistently used all our available disk space. It also appeared that some of the intermediate files hit the two gigabyte file limit on SAS databases in this release (6.11 TSO040).

  Eventually I created a file containing unique event numbers from the smaller database, and from that created a format containing about 600,000 items. The format mapped each event number to 'Y'. Then, in a data step over the large file, I deleted every record which the format didn't return a 'Y' for. SQL joins on the remaining file with the original smaller database were simple and used about 100 megabytes of disk

- I needed to read two very large (multi-million record) databases using FTP. They are to form one dataset on the SUN. Each record has the same record structure.

  I decided to read the files in one datastep, using the following structure:

  ```
  filename filea ftp .... ;
  filename fileb ftp .... ;

  data <whatever> ;
  infile filea ;
  ```

```
loop through filea until all read

infile fileb ;
loop through fileb until all read
run;
```

By doing this, I would have created one file immediately. Using two datasteps would have pushed disk space to the limit, as SAS would have needed to copy one of the datasets, thus doubling one in size for a period.

The code did not work. Because the infile statements open each file at compile time, the length of time taken to read the first file caused the second ftp link to time out.

A simple fix gets round this (thanks to Laurie Fleming for this suggestion). Each loop becomes a view:

```
data filea /view =filea ;
  infile filea ;
  read file a ;
run ;

data fileb /view =fileb ;
  infile fileb ;
  read file b ;
run ;
```

Then, run a datastep as follows:

```
data <whatever> ;
 set filea
     fileb ;
run ;
```

SAS does not request a view to execute its inherent datastep code until it is required. The INFILE opens get done at runtime when the data is actually required. Hence, the ftp does not time out, because it is not opened until required. The requirement of reading the data just once and storing as a SAS database is met using the views exactly as it would have been had the loop method worked.

Summary

If you have a resource hungry job, particularly when the resource is disk, you may find that some or all of ftp, views, and summary processing helps considerably. In particular,

- FTP removes the need to store remote input files locally
- views remove the need to create SAS databases to feed into SAS procedures
- SUMMARY processes classification groups in memory, so is likely to be more efficient than equivalent SQL processes

Additionally, there may be cases when the use of formats to reduce large databases before merging causes the merge to run much more successfully then it would using the whole database and relying on a WHERE or ON clause.