Conversion Methods From Oracle Tables to SAS Data Sets Using SAS/ACCESS, SAS Macro Language, and The UNIX Shell Script

Alex Gaber, Long Ngo
Ischemia Research and Education Foundation,
San Francisco, CA

1.0 Abstract

Massive clinical research data are stored in HP-UNIX Oracle 7.1.6 with SQL/NET V2 tables and managed by Oracle programmers who are unfamiliar with the SAS system. The data need to be converted to SAS data sets for analysis and presentation. Depending on the situation, there are different methods of conversion. Different alternatives were investigated and three methods were compared. This paper demonstrates the following three conversion methods investigated by a SAS programmer with some knowledge of the SQL language and UNIX shell script :

- Using Base SAS , Oracle SQL/Plus, and HP-UNIX shell script.
- Using the available SAS/ACCESS module in menu-driven mode.
- Using the available SAS/ACCESS module in a batch mode.

For each of the above three methods of data conversion, the advantages and disadvantages of each method will be pointed out. The SAS code and the UNIX Shell script will also be presented as well as relevant features of SAS/ACCESS and SQL. A real case study will be used throughout as an example. No knowledge of ORACLE is assumed for the readers. The goal of the paper is to introduce the users to some methods of accessing and converting data from Oracle to SAS depending on the availability of resources and expertise.

2.0 Case Study Data

The following data are part of a large clinical research data base stored in the relational data base system Oracle 7.1.6 in HP-UNIX 10.0. The data have been entered through an Oracle menu-driven data entry system. The data describe the electrocardiograph assessment done in each period on each of the patients in the study. The objective is to port the data to a SAS data set for statistical analysis. The data stored in Oracle are as follows :

Table 1 - Oracle Data

STUDY_ID (varchar2(10))	PATIENT_ID (varchar2(10))	ECG_PERIOD (varchar2(10))	ECG_DATE_TIME (date)
PROXIMATI3	801001	ICU	05-JUL-94:07:12
PROXIMATI3	801001	POD1	06-JUL-94:09:01
PROXIMATI3	801001	POD2	07-JUL-94:06:01
PROXIMATI3	801001	POD3	08-JUL-94:11:01
PROXIMATI3	801001	POD4	09-JUL-94:09:01
PROXIMATI3	801001	POD5	10-JUL-94:07:12
PROXIMATI3	801002	SCREEN	10-JUN-94:12:01
PROXIMATI3	801002	ICU	08-JUL-94:11:18
PROXIMATI3	801002	POD1	09-JUL-94:08:01
PROXIMATI3	801002	POD2	10-JUL-94:08:01

The Oracle data type for variables STUDY_ID, PATIENT_ID, and ECG_PERIOD is VARCHAR2(10) indicating that these variables are stored in Oracle as character type with maximum length of 10 spaces. This is equivalent to a SAS format of $10. For the variable ECG_DATE_TIME , the data type DATE is equivalent to DATEw. format in SAS.

To better understand the conversion process from Oracle to SAS, it is helpful to be familiar with Oracle terminology and its basic data base structure. First of all, the equivalence in terminology for some of the main features between an Oracle table and a SAS data set is described in the following table :

Table 2

SAS Terminology	Oracle Terminology	Note
Data Set	Table	
Variable	Column Name	
Observation	Record	
Format, Informat	Not Applicable	In Oracle, there is no equivalence. The data are stored as nonformatted values.
Length	Not Applicable	In Oracle, the length of the variable is defined by the system through the assigned Oracle data type whereas in SAS the length refers to storage length.
Label	Not Applicable	In Oracle the column name has a maximum length of 30 characters which also serve as the variable's label.

The objective is to convert the Oracle data (table 1) to a SAS data set shown in table 3 below :

Table 3 - Converted SAS data set

V1 ($10) LABEL= 'STUDY_ID(V1)'	V2 ($10) LABEL= 'PATIENT_ID (V2)'	V3 ($10) LABEL= 'ECG_PERIOD (V3)'	V4 (datetime13.) LABEL= 'ECG_DATE_TIME(V4)'
PROXIMATI3	801001	ICU	05JUL94:07:12
PROXIMATI3	801001	POD1	06JUL94:09:01
PROXIMATI3	801001	POD2	07JUL94:06:01
PROXIMATI3	801001	POD3	08JUL94:11:01
PROXIMATI3	801001	POD4	09JUL94:09:01
PROXIMATI3	801001	POD5	10JUL94:07:12
PROXIMATI3	801002	SCREEN	10JUN94:12:01
PROXIMATI3	801002	ICU	08JUL94:11:18
PROXIMATI3	801002	POD1	09JUL94:08:01
PROXIMATI3	801002	POD2	10JUL94:08:01

The format of the variable is indicated in the parentheses below the desired SAS variable name. Notice that the label for the SAS variable is from the Oracle table column's name. Within the label the SAS variable name is also embedded (e.g. 'study_id(V1)').

3.0 Method 1

3.1 Description

This conversion method illustrates how to convert Oracle data to SAS data set using only base SAS language, Oracle SQL language, and UNIX shell script commands. There are essentially three steps all of which are automated by the UNIX shell script program. First, the shell script program generates and executes an SQL program which takes as input the targeted Oracle table. The output from this SQL program is a flat file which contains the structure of the Oracle table to be converted. For the data above , the output which describes the structure of the Oracle table is stored in a flat file called ORDER.STR shown in **table 4** below.

Table 4 - ORDER.STR

```
L:43
S:ecg
T:ecg
```

```
N:4
V:STUDY_ID:V1:VARCHAR2:10:
V:PATIENT_ID:V2:VARCHAR2:10:
V:ECG_PERIOD:V3:VARCHAR2:10:
V:ECG_DATE_TIME:V4:DATE:13:
E:
```

The flat file ORDER.STR contains the description of the maximum length (L:43), source table (S:ecg), target name of SAS data set (T:ecg), number of variables (N:4), V is the prefix for the converted variables in sequential order from 1 to N (here V1 to V4), the Oracle column name and data type and length are also obtained showing for example the length of column name PATIENT_ID is 10 characters and in the SAS data set it will be converted to variable name V2.

The shell script program to generate ORDER.STR is shown in the appendix , part 1.

Secondly, ORDER.STR is used as an input file in the shell script program to create an SQL program ORDER.SQL shown in **table 5** below. Part 2 of the appendix shows the shell script program which generates ORDER.SQL.

Table 5 - ORDER.SQL

```
-- do not display commands from a file
set termout off;
set echo off;
-- no lines between pages
set newpage 0;
-- number of spaces between columns
set space 0;
-- no margins
set pagesize 0;
-- do not show the number of records returned by a query
set feedback off;
-- no headings
set heading off;
-- default width for numbers
set numwidth 20;
-- start spooling to the file order.dat
spool order.dat;
-- records length
set linesize  43 ;
-- default width for date
column ECG_DATE_TIME format a13;
select
STUDY_ID    V1
, PATIENT_ID    V2
, ECG_PERIOD   V3
,to_char(ECG_DATE_TIME,'ddmonyy:hh:mm') ECG_DATE_TIME
from  ecg ;
-- stops spooling
spool off;
exit;
```

Notice the SQL statement converting the column name ECG_DATE_TIME from oracle date type to SAS datetime13. format.
The shell script then executes this SQL program ORDER.SQL to extract the data from the Oracle table ECG. The data are output into the file ORDER.DAT which is shown in **table 6**.

Table 6 - ORDER.DAT

```
PROXIMATI3801001     ICU         05jul94:07:12
PROXIMATI3801001     POD1        06jul94:09:01
PROXIMATI3801001     POD2        07jul94:06:01
PROXIMATI3801001     POD3        08jul94:11:01
PROXIMATI3801001     POD4        09jul94:09:01
PROXIMATI3801001     POD5        10jul94:07:12
PROXIMATI3801002     SCREEN      10jun94:12:01
PROXIMATI3801002     ICU         08jul94:11:18
PROXIMATI3801002     POD1        09jul94:09:01
PROXIMATI3801002     POD2        10jul94:06:01
```

The third step is to have the shell script generate the SAS program to read in ORDER.DAT and create the SAS data set ECG.SSD. Part 3 of the appendix shows this section of the shell script generating and executing automatically the SAS program which is shown below in **table 7.**

Table 7 - ORDER.SAS

```
libname library '/users/analysis/aig/oracle/sas1/basessd';
libname epi2 '/users/analysis/aig/oracle/sas1/basessd';
data epi2.ecg (label='ecg');
infile '/users/analysis/aig/oracle/sas1/order.dat' linesize= 43  lrecl= 43  ;
input
V1   $   1 - 10
V2   $   11 - 20
V3   $   21 - 30
V4       datetime13.
;
label V1='STUDY_ID(V1)';
label V2='PATIENT_ID(V2)';
label V3='ECG_PERIOD(V3)';
label V4='ECG_DATE_TIME(V4)';
format  V4  datetime13.;
run;
proc contents data=_last_; run; proc print data=_last_ (obs=10)  labels; run;
```

Notice that the SAS program contains as input the Oracle column names' characteristics which are used in the SAS's INPUT statement. The options LINESIZE and LRECL also have the value of maximum record length 43 from the file ORDER.STR. The input uses the column format with column values also derived from ORDER.STR, the file containing the structure of the table. The shell script also executes this SAS program and creates the desired SAS data set shown in table 3.

3.2 Disadvantages

This conversion method requires some knowledge of Oracle SQL and UNIX shell script programming language. Additional temporary storage has to be available for the intermediary flat files. The procedure could also be time-consuming due to the additional disk I/O operations resulted from the creation of the temporary flat files.

3.3 Advantages

This procedure has five distinct advantages. First, it provides flexibility as to how the variable names and characteristics could be customized. Instead of the automatic generation of the sequential variable names given a prefix (e.g. with prefix V, V1 to V5), the program could be modified to have the desired variable names. Second, the created flat files provide wider accessibility if the data, instead of read into SAS, are to be read into some other software such as Excel, Paradox, or word processors ... all of which can read ASCII flat files. Third, this method requires only the basic tools which are base SAS, SQL which comes as basic component of Oracle, and the built-in UNIX shell script programming language. As a result, the four advantage is cost-effectiveness since tools such as SAS/ACCESS might not be readily available to some segment of users. And lastly the fifth advantage is that the conceptual model of this method could be easily applied to an environment where for example in DOS, the batch programming language could be used in place of the UNIX shell script language; furthermore, SQL is a standard tool and native to most relational data base system (RDBS); and base SAS is virtually compatible to all operating system platforms. This method therefore could be easily applied to an environment running Windows '95, Paradox, and Windows SAS system.

4.0 Method 2

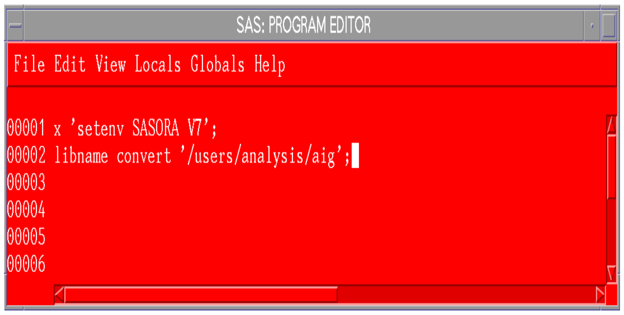
4.1 Description

If SAS/ACCESS is available then there is another method of converting Oracle table data to SAS data set. Below is the step-by-step procedure that the user could go through systematically to create a SAS data set from an Oracle table :

- First of all, Oracle client software has to be preinstalled on the UNIX workstation. The Oracle data base administrator (DBA) has to assign access privilege.

The DBA has to give the SAS user a user name, password, location or path of access to the specific Oracle table to be accessed by SAS. Then a link between Oracle and SAS/ACCESS has to be established . This step requires that the UNIX environment variable SASORA be created. One can edit the system file .cshrc to add the command 'setenv SASORA V7', or one can just issue and submit the shell command 'x setenv SASORA V7' from within the SAS program editor. Figure 1 shows the X shell command and the creation of the location in which the SAS/ACCESS-created files will be stored (libref CONVERT).

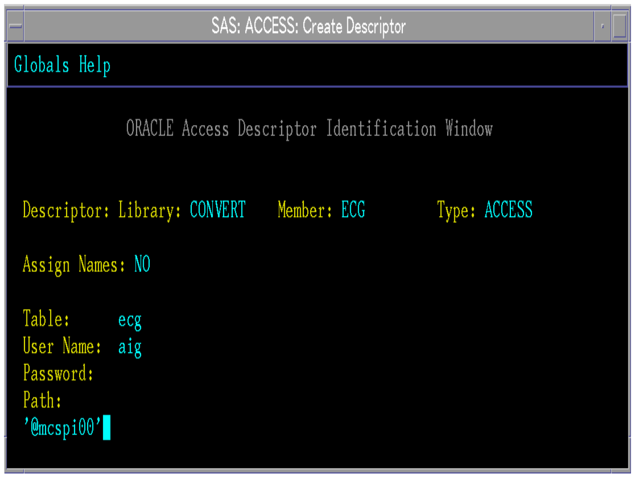
Figure 1



```
x 'setenv SASORA V7';
libname convert '/users/analysis/aig';
```

- Create an object called access descriptor. This step could be done by going to the ACCESS option under the GLOBAL menu (global, access, file, new, libref name). Figure 2 shows the Oracle Access Descriptor Identification Window in which the user has to input MEMBER which is the name of the object descriptor in libref CONVERT. The value NO on the field ASSIGN NAME means that no default SAS variables names be created because we want the variables to be V1 to V4. The field TABLE has the filled-in value of ECG which is the name of the ORACLE table to be converted. USER NAME, PASSWORD, and PATH are the DBA-assigned information that the user must have for these fields.

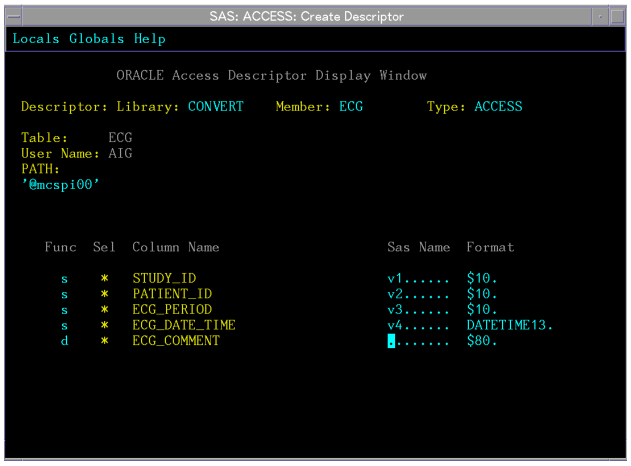
Figure 2



```
          ORACLE Access Descriptor Identification Window

Descriptor: Library: CONVERT   Member: ECG        Type: ACCESS

Assign Names: NO

Table:     ecg
User Name: aig
Password:
Path:
'@mcspi00'
```

- This step shows the window containing the information of the Oracle table ECG from which the user now must enter the SAS variable names under SAS NAME and the corresponding formats. Notice from figure 3 that only the first four wanted variables were selected. When this step is finished, a file called ECG.SSA01 will be created in CONVERT represent the access descriptor object.

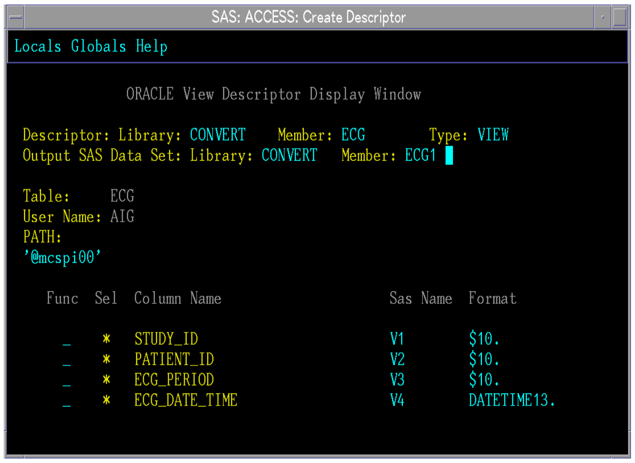
Figure 3



```
             ORACLE Access Descriptor Display Window

Descriptor: Library: CONVERT    Member: ECG        Type: ACCESS

Table:     ECG
User Name: AIG
PATH:
'@mcspi00'


   Func  Sel  Column Name                      Sas Name  Format

     s    *   STUDY_ID                         v1......  $10.
     s    *   PATIENT_ID                       v2......  $10.
     s    *   ECG_PERIOD                       v3......  $10.
     s    *   ECG_DATE_TIME                    v4......  DATETIME13.
     d    *   ECG_COMMENT                      ........  $80.
```

- Figure 4 shows the final step which is the View Descriptor Display Window showing the resulted SAS names and formats. The output SAS data set can be created here by entering the location for the library (in this case CONVERT) and the name of the data set which is required to be different from the view descriptor object name (ECG). In CONVERT the view descriptor object is stored in the file ECG.SSV01. The SAS name in this case is ECG1 and the physical name is ECG1.SSD01.

Figure 4



```
             ORACLE View Descriptor Display Window

Descriptor: Library: CONVERT    Member: ECG        Type: VIEW
Output SAS Data Set: Library: CONVERT   Member: ECG1

Table:     ECG
User Name: AIG
PATH:
'@mcspi00'

   Func  Sel  Column Name                      Sas Name  Format

     _    *   STUDY_ID                         V1        $10.
     _    *   PATIENT_ID                       V2        $10.
     _    *   ECG_PERIOD                       V3        $10.
     _    *   ECG_DATE_TIME                    V4        DATETIME13.
```

4.2 Disadvantages

- Customized variable names and their formats have to be manually entered in the ACCESS descriptor window.
- Software availability and support of SAS/ACCESS module and Oracle client software are required.

4.3 Advantages

- This method is very user-friendly due to SAS/ACCESS menu-driven conversion procedure.

- Once the view descriptor object has been created, the Oracle data could be accessed dynamically by SAS software so that the whole conversion procedure does not need to be repeated entirely even though the Oracle table data have been updated.
- With the availability of SAS/ACCESS and Oracle client software, no other tools are needed so when the entire procedure is set up, it is relatively easy to maintain and to train the users.

5.0 Method 3

5.1 Description

This method of conversion is similar to method 2. However, instead of using the SAS/ACCESS menu-driven environment, the user could use batch mode to carry out the same operations. The idea is the access descriptor object and the view descriptor object can be created in a SAS program using Proc Access directly. Table 8 below shows the SAS code for creating the access descriptor.

Table 8

```
libname convert  "/users/analysis/aig";
X 'setenv SASORA V7';
* define  SAS/Access descriptor for the  ORACLE table to be converted *;
proc access dbms=oracle;
 create convert.ecg.access;      /*create the access descriptor object */
 user=aig;                        /*user name given by Oracle DBA   */
 orapw=xxxxxxx;                     /*user password given by DBA  */
 table=ecg;                       /*Oracle table name to be converted */
 path='@mcspi00';                    /*location of the Oracle table  */
 list all;
run;
```

Notice that all the information needed to access the Oracle table have to entered into the program just like they were required in method 2 using the menu-driven SAS/ACCESS window environment.

Next, the view descriptor object has to be created. Then the SAS data set can be created from the view descriptor object. Table 9 shows the SAS code for creating both the view descriptor and the SAS data set.

Table 9

```
* define the ecg table view descriptor - rename variables to valid sas names. *;
proc access dbms=oracle accdesc=convert.ecg;
/* call the access descriptor object */
 create convert.ecg.view;
 /* create the view descriptor object*/
 select study_id patient_id ecg_period ecg_date_time;
 /* selected wanted variables*/
 rename study_id=v1 patient_id=v2 ecg_period=v3 ecg_date_time=v4;
/*rename */
 list view;
run;
* creating the SAS data set from the view descriptor object;
data convert.ecg1;
 set convert.ecg;
 format v1-v3 $10. v4 datetime13.;
run;
```

5.2 Disadvantages

For this method, the disadvantages are the similar as in method 2.
- Customized variable names and their formats have to be manually written into the SAS program.
- Software availability and support of SAS/ACCESS module and Oracle client software are required.

5.3 Advantages

Similar to method 2, this procedure allows the user to use only SAS tools to convert Oracle data into SAS. Once the view descriptor has been created, SAS software can be used directly to access Oracle table even though the Oracle table data have been updated. One advantage of this method that method 2 lacks is that this method allows the user to customize the conversion procedure by, for example, taking as an input the name of an Oracle table and pass it to the conversion programs as macro input parameter. This method also documents the conversion steps in the programming statements.

6.0 Conclusion

Essentially the most important difference which separates the three conversion methods above is the availability of the module SAS/ACCESS. Without SAS/ACCESS, a great deal of effort is needed to develop customized conversion programs using tools such as UNIX shell script and SQL programming languages as shown in method 1. However, once developed this method 1 procedure allows a great deal of flexibility such as automating sequential variable names which requires only the specification of the prefix (e.g. V as in above). Some expertise in tools such as SQL and shell script are also required to access and retrieve the Oracle data. Method 2 and 3 are much more user-friendly but SAS/ACCESS module must be available and Oracle client software must be installed on the user's machine. Once setup this procedure allows the user to use only the SAS/ACCESS software to access and retrieve Oracle data.

7.0 Reference

SAS/ACCESS® Interface to ORACLE®: Usage and Reference, Version 6, Second Edition

SQL*Plus® User's Guide and Reference, ORACLE®

HP-UX Reference, HP 9000 Computers, HEWLETT PACKARD®

8.0 Authors address

Alex Gaber, M.S.
Long Ngo, Ph.D.Cand.
Ischemia Research and Education Foundation
250 Executive Park Blvd. #3400
San Francisco, CA 94134-3306
(415) 715-2319

APPENDIX

```
# .ORDER.ORDER:
#
#!/bin/csh

echo "--------------------------------------" >>order.track
echo "EXTRACTION PROTOCOL" >> order.track
id >> order.track
pwd >> order.track
echo "ORACLE TABLE: $1 " >> order.track
echo "SAS DATA SET: $2 " >> order.track
echo "START TIME: " >> order.track
date >> order.track

# keep table name in ORDER.NAM:
echo "S:$2" >> order.nam
echo "T:$1" >> order.nam

# PART 1: GENERATE ORDER.STR

# create sql order.sql and ask ORACLE to run it:
echo 'set termout off;' > order.sql
echo 'set echo off;' >> order.sql
echo 'set newpage 0;' >> order.sql
echo 'set space 0;' >> order.sql
echo 'set pagesize 0;' >> order.sql
echo 'set feedback off;' >> order.sql
echo 'set heading off;'  >> order.sql
echo 'set numwidth 80;'  >> order.sql
echo 'spool order.lst;' >> order.sql
echo "describe $1 ;" >> order.sql
echo 'spool off;' >> order.sql
echo 'exit' >> order.sql
sqlplus -s aig/aig@mcspi00 @order

# to get the structure of table into ORDER.VAR:
awk -f .order.awk < order.lst > order.var
rm order.lst
rm order.sql

# create structure file ORDER.STR
sed "s/ //g" < order.var > order.tmp
wc -l order.tmp | awk '{print "N:",$1-1}' | sed "s/ //g" > order.var.number
awk -f .orderlength.awk < order.var | sed "s/ //g" > order.tmp1
cat order.tmp1 order.nam order.var.number order.tmp > .order.str
rm order.tmp order.tmp1 order.nam order.var order.var.number
```

# PART 2: GENERATE ORDER.SQL

```
# to get ORACLE table values into ORDER.DAT:
awk -f .orderdata.awk < .order.str > order.tmp
awk -f .orderdata1.awk < .order.str > order.tmp1
awk -f .orderdata2.awk < .order.str > order.tmp2
cat order.tmp order.tmp1 order.tmp2 > order.sql
rm order.tmp order.tmp1 order.tmp2
sqlplus -s aig/aig@mcspi00 @order
rm order.sql
```

# PART 3: GENERATE ORDER.SAS

```
# to create SAS program ORDER.SAS
awk -f .ordersas.awk < .order.str > order1.sas
awk -f .ordersas1.awk < .order.str > order2.sas
cat order1.sas order2.sas > order.sas
rm order1.sas order2.sas

# run ORDER.SAS
sas order

echo "FINISH TIME: " >> order.track
date >> order.track
#awk -f .orderbyt.awk < .order.str > order.tmp1
#cat order.track order.tmp1 > order.tmp2
#cp order.tmp2 order.track
#rm order.tmp1 order.tmp2
#echo "--------------------------------------" >>order.track

grep -i error order.log >> order.track
if ($status == 0) then
   echo "SOMETHING WRONG WITH EXTRACTION DATA $2 FROM $1" >>
order.track
   echo "Please call Alex Gaber" >> order.track
else
endif
grep -i warning order.log >> order.track
if ($status == 0) then
   echo "SOMETHING WRONG WITH EXTRACTION DATA $2 FROM $1" >>
order.track
   echo "Please call Alex Gaber" >> order.track
else
endif
grep -i note order.log >> order.track
order.track
echo "--------------------------------------" >>order.track
echo "--------------------------------------" >>order.track

#cleaning
#rm order.dat .order.str
#rm order.log order.lst order.sas

#END
```

######### SUBPROGRAMS ############:

# .**ORDER.AWK**

```
#
BEGIN {
FS=",[ \t]*|[ \t]+";
sasv=0;
}
{
$0=substr($0,2);
index_type=index($0," VARCHAR2");
if (index_type>0) {
   ++sasv;
   var_name=substr($0,1,(index($0," ")-1));
   sas_var_name="V" sasv;
   var_type=substr($0,index_type);
var_length=substr(var_type,(index(var_type,"("))+1,(index(var_type,")"))-
(index(var_type,"("))-1);
   if  ((length(var_length)  >=  3)  &&  (substr(var_length,1,1)  >  "1"))  {
var_length=200; }
   var_type=substr(var_type,1,(index(var_type,"(")-1));
   print "V:",var_name,":",sas_var_name,":",var_type,":",var_length,":";
   }
index_type=index($0," DATE");
if (index_type>0) {
   ++sasv;
   var_name=substr($0,1,(index($0," ")-1));
   sas_var_name="V" sasv;
   var_type=substr($0,index_type);
   var_length="13";
   print "V:",var_name,":",sas_var_name,":",var_type,":",var_length,":";
   }
index_type=index($0," NUMBER");
if (index_type>0) {
   ++sasv;
   var_name=substr($0,1,(index($0," ")-1));
   sas_var_name="V" sasv;
   var_type=substr($0,index_type);
  (index(var_type,"(")-1))+1;
   var_length=20;
   var_type=" NUMBER";
   print "V:",var_name,":",sas_var_name,":",var_type,":",var_length,":";
   }
}
END {
print "E:";
}
```

#.**ORDERLENGTH.AWK**

```
#
BEGIN {
FS=":";
i=0;
}
{
i += $5;
}
END {
print "L:",i
}
```

#.**ORDERDATA.AWK**

```
#
BEGIN {
FS=":";
print "set termout off;";
print "set echo off;";
print "set newpage 0;";
print "set space 0;";
print "set pagesize 0;";
print "set feedback off;";
print "set heading off;";
print "set numwidth 20;";
print "set arraysize 1;";
print "spool order.dat;";
}
{
if ($1=="L") {
   print "set linesize ",$2,";";
   }
if ($1=="V") {
   if ($4=="DATE") {
       printf ("column %s format a%s;\n",$2,$5);
       }
if ($4=="VARCHAR2") {
       printf ("column %s format a%s;\n",$3,$5);
       }
   }
}
END {
}
```

#.**ORDERDATA1.AWK**

```
#
BEGIN {
FS=":";
i=0;
}
{
if ($1=="N")
   {
   print "select ";
   }
if ($1=="V")
   {
   if (i==0)
      {
      if ($4=="DATE")
         {
         printf ("to_char(%s,'ddmonyy:hh:mm') %s\n",$2,$2);
         }
      else
         {
         print $2," ",$3;
         }
      }
   else
      {
      if ($4=="DATE")
         {
         printf (",to_char(%s,'ddmonyy:hh:mm') %s\n",$2,$2);
         }
      else
         {
         print ",",$2," ",$3
         }
      }
   i=1;
   }
}
END {
}
```

#.**ORDERDATA2.AWK**

```
BEGIN {
FS=":";
}
{
if ($1=="T")
   {
   print "from ",$2,";";
   }
}
END {
print "spool off;";
print "exit;";
}
```

#.ORDERSAS.**AWK**

```
#
BEGIN {
FS=":";
print "libname library '/users/analysis/aig/oracle/sas1/basessd';";
print "libname epi2 '/users/analysis/aig/oracle/sas1/basessd';";
l=1;
s=0;
f=0;
}
{
if ($1=="L") { lrecl=$2; }
if ($1=="S") { dataset=$2; }
if ($1=="T") {
   tablename=$2;
   printf ( "data epi2.%s (label='%s'); \n",dataset,tablename);
   }
if ($1=="N") {
   print "infile '/users/analysis/aig/oracle/sas1/order.dat' linesize=",lrecl,"
lrecl=",lrecl," ;";
   print "input";
   }
if ($1=="V") {
   s = s + l ;
   f = s + $5 -1 ;
   l =$5 ;
   if ($4=="NUMBER") {type=" ";  print $3," ",type," ",s,"-",f; }
   if ($4=="VARCHAR2") {type="$";  print $3," ",type," ",s,"-",f; }
   if ($4=="DATE") {type=" ";  print $3," ",type," ","datetime13."; }
   }
if ($1=="E") {
   print ";";
   }
}
END {
}
```

#**.ORDERSAS1.AWK**

```
#
BEGIN {
FS=":";
}
{
if ($1=="V") {
   printf ("label %s='%s(%s)'; \n",$3,$2,$3);
   if ($4=="DATE") { print "format ",$3," datetime13.; "; }
   }
if ($1=="E") {
   print "run;";
   print "proc contents data=_last_; run; proc print data=_last_ (obs=10)  label;
run;";
   }
}
END {
}
```

#**.ORDERBYT.AWK**

```
#
BEGIN {
FS=":";
l=0;
n=0;
}
{
if ($1=="L") { l=$2; }
if ($1=="N") { n=$2; }
}
END {
printf ("NUMBER OF VARIABLES: %d, RECORD LENGTH: %d, BYTES:
%d\n",n,l,n*l);
}
```