

Using SAS Views to Enhance Software Quality, Improve Data Security and Reduce Data Replication

by Adam Hendricks
Clinical Programmer/Analyst
ICOS Corporation

Sometimes it seems that the strengths of the SAS language are also its weaknesses.

1. It allows for many ways to accomplish the same task.
2. It allows for a wide range of technical skill levels.
3. Its datasets are unique OS files that are relatively easy to access and manage.

Strengths

SAS is popular with analysts for the following reasons and more:

- It does not require one to spend too much time studying programming or database design.
- Compared to other languages, SAS requires less dependence on professional programmers to get tasks completed.
- It has a wide selection of reliable statistical and reporting procedures that are fairly easy to use.

SAS is popular with programmers for the following reasons and more:

- It is a very powerful data manipulation and analysis tool.
- It is easier to deal with than third generation languages such as COBOL, Pascal, FORTRAN, or C.
- It is a cheaper and easier way to establish a high performance database than most RDBMS software.
- It is possible to develop useful applications very rapidly.
- It has an SQL procedure.

Weaknesses

- SAS is too flexible. It is very difficult to establish coding standards in a language that offers up so many different ways to get the same task accomplished (for example frequency counts, mean calculation, or extracting subsets of data). If many of the SAS scripts used at a single site contain data setup routines meant to perform the same task but written by different users, then it can be difficult to ensure consistent data setup across all scripts at the site. It is difficult to establish standard SAS scripts in busy work environments because anyone who can run a standard script can copy it into their home directory, alter it and then pass it on to another user.
- Efficient data storage can make data useless to end users. Establishing normal forms in a SAS database can be difficult if many of the end users are novice programmers and therefore do not have the skills to manipulate data properly for analysis and reporting purposes. In order to make normalized source datasets more usable to novice programmers, denormalized analysis datasets can be generated. This is a duplication of data that must be repeated every time the source data changes. If the source data changes often and analyses are run frequently, then keeping analysis datasets up to date could be unmanageable.
- Complex data setup routines for procedure input data can become too inefficient and possibly erroneous if setup tasks are left up to novice programmers.
- Source data that is contained on a non-secure disk area with no password protection can be accidentally deleted or altered.
- Confidential data that is not password protected is available to anyone with OS read access to the dataset. Applying password protection to a dataset containing confidential information could block off access to useful but non-confidential information contained in the same dataset.
- SAS datasets from data entry applications, e.g. SAS/FSP or Clintrial, can contain fields that are useful to data management personnel but useless and confusing to some end users. Removing the unwanted

fields may not be practical and could require duplication of source SAS data if FSP is the data entry system.

The weaknesses mentioned here can be dealt with in several ways:

- Limiting access to SAS code and data on an operating system level. This could slow down the data analysis process and annoy end users.
- Imposing more rigid controls on SAS code development through more management oversight. How often does anything good come from more management oversight of a technical issue?.
- Try using SAS views.

General Description of SAS Views

SAS views are two of the three kinds of compiled SAS code:

1. The stored program facility (SAS Language Reference, v. 6, First Edition, Appendix 3, p. 983)
2. Data step view (SAS Language Reference, v. 6, First Edition, Ch. 6, p. 200 and SAS Technical Report P-222, ch. 14, p. 149)
3. SQL view (SAS Guide to the SQL Procedure, v. 6, First Edition, Ch. 5, p. 99)

Stored programs are not views because stored programs produce SAS datasets; views do not produce datasets. It is also possible to store compiled SAS macro code in a catalog, but compiled macro code is not the same as compiled SAS code. Macro code is normally used to generate SAS code which is then sent to the SAS compiler. Stored SAS code is already compiled.

Views are used in code as if they were actual SAS datasets. There are two engines associated with SAS views:

1. SASDSV - SAS data step view which is a stored data step. Generating a data step view is similar to generating a stored program (SASPGM engine). The VIEW= option is used at compilation time to create a view rather than the PGM= option which is used to create a stored program. Possible data flows for data step views on UNIX are described in Figure 1 (All figures are located in the last five pages of this document.). A data step view does not store its source code internally. It must be saved elsewhere.
2. SASESQL - PROC SQL view which is a stored SELECT statement. Generating a SQL view is similar to generating a table. CREATE VIEW is used to generate a view, and CREATE TABLE is used to generate a SAS dataset. Possible data flows for data step views on UNIX are described in Figure 2. An SQL view stores its source code internally. The source code, except for passwords, can be printed to the log using the DESCRIBE VIEW <View Name> command.

Although a view can be used in code just as a SAS dataset would be used. It is different in the following ways:

- A view does not contain any data. It is stored code that processes and channels data from a dataset or another view to any data step or procedure that references the view. PROC CONTENTS can be run on a view, but the observations will be missing.
- Execution error checking is not done at the time the view is compiled. It is done when the view is referenced. Only syntax is checked during view compilation. The view code is not compiled to machine code or executed on the host until it is referenced by a procedure or data step requesting data. This can cause unusual log messages (e.g. PROC PRINT can generate log messages from one or more data steps or SQL select statements if it is referencing a view.).
- SASDSV single pass views do not create work files. A single pass view will process and feed one observation at a time to the requesting data step or procedure. This will therefore decrease I/O and run time. SASDSV multiple pass views (views that use BY groups) use an external spill file that holds one BY group at a time. The spill file never gets larger than the largest BY group, and it reclaims

work space. If there are many BY groups of significant size contained in the source data, then a view will use much less temporary storage than an intermediate work dataset.

- Processing data with views uses a slightly different set of available options and attributes than processing data with the default engine using data steps or CREATE TABLE statements. See Table 1.

Table 1. Differences between View Engines and Default Engines for Processing

Available Option or Attribute	Default Using SQL	Default Using Data step	SAS/SQL	SAS/DSV
PW= Data Step Option	X	X	X	X
READ= "	X	X	X	X
WRITE= "	X	X	X	X
ALTER= "	X	X	X	X
INDEX= "	X	X		
SORTEDBY= "		X		X
LABEL= "	X	X		X
DROP= "	X	X		X
KEEP= "	X	X		X
TYPE= "	X	X		X
RENAME= Dataset Option or Col. Alias	X	X	X	X
Sort Data in Same Step	X		X	
Can process obs. one by one in sort order		X		X
Can Use Aggregate Functions	X		X	
Allows Non-SAS Data Sources		X		X
If-Then Processing	X	X	X	X
Implicit Num./Char. Conversion		X		X
Reference Implicitly from a PROC			X	X
Source Code Accessible from Object	n/a	n/a	X	
Source Code Must Be Accessible to User	X	X		

Enhancing Software Quality

Guaranteeing SAS software quality is difficult if all processing is done with SAS scripts run by the users. If a user can run the script, then he can copy it, alter it, and use the copy to accomplish the task for which the original code was intended. This can undermine the software quality process. Over time scripts can degrade due to "fixes" implemented by inexperienced programmers to deal with problems that may have nothing to do with the SAS script itself. But may be due to a problem with the source data quality or to a bad assumption made by the programmer about the source data.

Using SAS views can enhance software quality by keeping critical SAS code secure from alteration. Coding standards can be enforced by limiting access to source data through the use of password protection of source data and open views to the source data. Open views are not password protected and should provide a means to access all necessary information from the source data. This takes the burden of writing complex setup code out of the hands of users who simply want to write a report or view a list. It also brings peace of mind to applications programmers who are responsible for ensuring software quality in a situation where many novice programmers need access to information produced by complex SAS processing.

Figure 3 shows how views can shape a normalized database into view schemas that are more usable to an analyst. Also, changes in the source data are immediately made available through a view accessing the changed data without having to rerun the view source code. Figure 4 demonstrates how a change in source data is propagated by views to the users who do not have direct access to the source datasets. The sample code in the appendix shows a series of views that work off the sample data supplied by SAS in release 6.11.

The source code for a SAS view need not be available to the user accessing the view. Although the source code, except passwords, for a SQL view can be written to the log using the DESCRIBE VIEW command. Passwords do not show up on the SAS log for obvious reasons.

Improving Data Security

Data security can be enhanced by making columns containing confidential information unavailable without having to block access to all columns in the same dataset. Figure 3 demonstrates this. All three source datasets - EMP, DEPT, and JOBS - contain confidential information and therefore can not be made available to all users. Two views are generated. One, V_OPEN, contains information that can be seen by all users, and the other, V_SECURE, contains information that can only be seen by users who have the appropriate password.

A SQL view can take advantage of aggregate functions to create an open, summary view of data without allowing the user to have access to confidential source data from which the summarized view is derived. Figure 5 demonstrates this. The secure view from figure 3, V_SECURE, containing individual salary information is used to generate an open view, V_MEANS, containing mean salary by department, sorted from high to low.

By allowing for enforced data setup strategies, the risk of misinformation being derived from a SAS database due to poor setup code is reduced.

Reducing Data Replication

If end users require denormalized data from a normalized SAS database or subsets of data from SAS datasets that need to be secure, then data must be replicated into datasets that meet both user and security requirements. Or a view must be used. Replicating data will increase the disk space required to store a SAS database. It also requires that the analysis files must be updated every time the source data is updated if the analysis datasets are to remain current. Views are always current at the time of execution since they only channel and process source data and do not replicate it to a separate, new dataset. Figure 3 shows two views that reference the same columns. But the data is still normalized and non-replicated even when the views seem to be duplicate, non-normal SAS data from a conceptual standpoint.

Working with Views

Setting up reporting applications using views has a different feel to it than developing SAS scripts. Here are the most noticeable changes:

- Views that reference or get referenced by other views do not have to be generated in a particular order. This is because there is no data being passed around at compile time. This is convenient because it allows view source code to be handled like modules. The view must be tested by being referenced by a data step or procedure in order to be fully debugged; since view code can compile without an error message but still generate errors at execution time.
- When a SQL view that uses a column in a GROUP BY statement and the column was derived from an earlier view incorporating an aggregate function, a column number alias can not be used in the second view. Here is an example:

```
* Generate quartile values from salary rank *;
proc sql;
  create view vsample.salrank as
  select a.empno, a.gender, a.salary, b.salrank,
         case
           when salrank<=0.25*max(salrank) then 1
           when 0.25*max(salrank)<salrank<=0.5*max(salrank) then 2
           when 0.5*max(salrank) <salrank<=0.75*max(salrank) then 3
           else 4
         end as quartile label='Quartile'
  from vsample.emp (pw=&mypw) a,
       vsample.payrank2 (pw=&mypw) b
  where a.salary = b.salary
  order by 4;
```

```

* Wrong! This will not work because QUARTILE is derived *;
* using an aggregate function which will be executed.   *;
* when this view is referenced.                         *;

```

```

create view vsample.qtile1 as
select quartile, gender,
       count(*) as count label='Count'
from vsample.salrank
group by 1, 2; * <-- problem is here. *;

```

```

* Correct. Use the alias or suffer. *;
create view vsample.qtile1 as
select quartile, gender,
       count(*) as count label='Count'
from vsample.salrank
group by quartile, gender; * <-- spell it out. *;

```

- PROC SQL is limited to 16 tables per query. A SQL view being referenced by a SQL query is not counted as a single table but as the total number of tables referenced by the SQL view. To prevent this from being a problem. View source code should read from actual data whenever possible. A view can reference a view, but care has to be taken to not go over the limit. A workaround for this problem is to use a data step view for operations that can't be handled by SQL due to the 16 table limit. A data step view is not burdened with such a low table limit. Here is an example of two views meant to perform the same task:

```

* This will work... *;
data a954iv.aettab11(pw=red)
    keep=bodysys term n0 p0 n2 p2 n4 p4 n22 p22
    label='AE Term Summary Setup I')/
view=a954iv.aettab11;
merge a954iv.aetrmsum(in=a where=(dosegrp=0)
    rename=(count=n0 percent=p0))
    a954iv.aetrmsum(in=b where=(dosegrp=2)
    rename=(count=n2 percent=p2))
    a954iv.aetrmsum(in=c where=(dosegrp=4)
    rename=(count=n4 percent=p4))
    a954iv.aetrmsum(in=d where=(dosegrp=22)
    rename=(count=n22 percent=p22));
by bodysys term;
if a and b and c and d;
run;

```

```

* ...where this will not if 'a954iv.aetrmsum' is a *;
* view that references more than 4 tables.           *;
proc sql;
create view a954iv.aettab11(pw=red) as
select a.bodysys, a.term,
       a.count as n0, a.percent as p0,
       b.count as n2, b.percent as p2,
       c.count as n4, c.percent as p4,
       d.count as n22, d.percent as p22
from a954iv.aetrmsum a,
     a954iv.aetrmsum b,
     a954iv.aetrmsum c,
     a954iv.aetrmsum d
where a.bodysys = b.bodysys = c.bodysys = d.bodysys and
     a.term    = b.term    = c.term    = d.term    and
     a.dosegrp = 0          and
     b.dosegrp = 2          and
     c.dosegrp = 4          and
     d.dosegrp = 22
order by 1, 2;

```

- Both of the last two code examples perform the same task, but the data step view is more efficient (as is almost always the case). SQL views are useful when a new sort order is needed, a summarized view is needed, or a merge of unsorted data is required.

Downside of Views

- View code is more difficult to debug than data steps or CREATE TABLE statements.
- Using SAS views to handle complex data setup tasks precludes the use of SAS procedures that can do the job better. For example, if a transpose is required for data setup, PROC TRANSPOSE is a better tool to use for transposing than a data step view or PROC SQL. But PROC TRANSPOSE generates a dataset and therefore must be, in some way, part of a SAS script run by the user. If the goal is to provide views that are ready to be used as input for reporting or analysis procedures contained in simple programs, then adding a TRANSPOSE step adds an extra level of complexity to the user-run scripts.
- Some complex setup tasks could be more easily handled with SAS macros - provided that security is not an issue.

Upside of Views

- A data setup system of views can provide, convenient, ready to use data for analysis and reporting tasks performed by novice programmers.
- Views can provide access to non-SAS files and OS command output as if each were a SAS dataset.
- Views can help keep data and setup routines secure and standardized without inconveniencing end users.
- Views can reduce overall user CPU time and work space usage in many situations.

Conclusions

Views definitely are worthwhile for situations where data security is an issue. For problems of standard software degradation, there are other methods available such as compiled macros or stored programs. As far as I know, compiled macros and stored programs are not intended to be used for software security but are, in fact, often used for just that purpose. It is probably easier for most experienced SAS programmers to use compiled macro code to keep software secure, but that may require using the compiled macro facility in the manner for which it was not intended. SQL views get my vote as the best tool for software security in SAS/BASE because the source code can be accessed from the object itself by using DESCRIBE VIEW which is a good backup in case the actual source code is lost.

I have to admit that I am guilty of pushing the limit a bit in testing the functionality of SAS views as demonstrated in the appendix, but it was fun to see what could be done. In the appendix I have provided examples of sample code that set up a series of views to sample data provided with the SAS 6.11 release. Included in the samples is a series of views designed to calculate ranks on employee salary levels and a view to run a Student's t-test on those ranks. While it is certainly preferable to use PROC RANK, PROC TTEST, or PROC NPAR1WAY for these purposes, the sample application views provide a means to perform these tasks without the generation of intermediate SAS datasets. This allows for an in-depth summarizing of the source data using nothing more PROC PRINT.

I won't say that views are panacea to whatever ails your SAS site, but SAS views the best data security tool available in SAS and when used properly can add new dimensions to a database without adding new SAS datasets or user-run SAS programs.

Appendix

```
* SAS View Samples
*
* SAS Version: 6.11 for Solaris I
*
* Program: viewdemo.sas
*
* Uses the following SAS supplied sample data:
*
*     /home2/sas611/samples/base/employee.sd01
*     /home2/sas611/samples/base/salary.sd01
*     /home2/sas611/samples/base/leave.sd01
*     /home2/sas611/samples/base/jobcodes.sd01
*
* Function: Copies the SAS supplied sample data to a local subdirectory,
*           password protects indexes the copied files. Generates a series
*           of sample views that can be used with the SAS sample data. This
*           program is UNIX specific. Notice that the views are not compiled
*           in the order that they will be referenced at execution time.
*
* Programmer: Adam Hendricks, ahendric@icos.com, (206)485-1900 x2295
*
* Date: 5/17/96
*
* Update:
*
* ;

* Assign the same password to all *;
* data and protected views for    *;
* simplicity.                      *;
%let mypw = nopeekin;

* SAS Sample Data *;
libname samples '/home2/sas611/samples/base';

* Local Demo Data and Views *;
libname vsample '$HOME/viewdemo/vsample';

* Remove Read-Only Protection for *;
* Rebuild of View Library.          *;
x 'chmod 644 /$HOME/viewdemo/vsample/*.*';

* Delete Old Data and Views *;
x 'rm /$HOME/viewdemo/vsample/*.*';

* Copy SAS Sample Data to Local View Demo Library *;
proc copy in=samples
          out=vsample
          memtype=data;
run;
```

```

* Add Password Protection to Source Data.      *;
* Add Simple Indexes to Source Data Columns    *;
* Used in EMP View Constructed for Demo Lib.   *;
proc datasets lib=vsample;
  modify employee(pw=&mypw);
    index create jobcode;
    index create idnum;

  modify jobcodes(pw=&mypw);
    index create jobcode;

  modify salary(pw=&mypw);
    index create idnum;

  modify leave(pw=&mypw);
    index create idnum;
run;

* Create SQL Views *;
proc sql;
  *****;
  *                                     *;
  * View Name: vsample.emp             *;
  *                                     *;
  * Protected: Yes                     *;
  *                                     *;
  * Engine: SASesQL                    *;
  *                                     *;
  * Uses: vsample.employee (data)      *;
  *       vsample.jobcodes (data)      *;
  *       vsample.salary (data)        *;
  *       vsample.leave (data)         *;
  *                                     *;
  * Sorted By: empno                   *;
  *                                     *;
  * Function: Confidential Employee Information *;
  *       Table.                       *;
  *                                     *;
  *****;
  create view vsample.emp(pw=&mypw) as
  select distinct
    a.empno    label='Employee Number',
    a.name     label='Last Name, First Name, MI',
    a.divcode  label='Division Number',
    a.deptcode label='Department Number',
    a.location label='Office Location',
    b.title    label='Job Title',
    a.room     label='Room Number',
    a.phone    label='Extension',

    /* Calculate Years Employed. */
    (date()-a.hdate)/365.25 as yrs
    format=4.1 label='Total Years Employed',

```

```

/* Calculate Days of Non-Vacation Leave. */
/* Set to Zero if None Taken. */
case d.lvenddte
  when . then 0
  else      sum(d.lvenddte-d.lvbegdte+1)
end as dayslv label='Total Days Leave',

/* Calculate Age. */
floor((date()-a.birthday)/365.25) as age label='Age',
a.gender label='Gender',
c.salary label='Salary'
from vsample.employee(pw=&mypw)      a
left join vsample.jobcodes(pw=&mypw) b
on a.jobcode = b.jobcode
left join vsample.salary(pw=&mypw)   c
on a.idnum  = c.idnum
left join vsample.leave(pw=&mypw)    d
on a.idnum  = d.idnum
group by 1;

*****;
*                                           *;
* View Name: vsample.empdir                *;
*                                           *;
* Protected: No                           *;
*                                           *;
* Engine: SASOQL                          *;
*                                           *;
* Uses: vsample.emp (view)                *;
*                                           *;
* Sorted By: name                         *;
*                                           *;
* Function: Open Employee Information Table *;
*                                           *;
*****;
create view vsample.empdir as
select name      label='Name',
       title     label='Job',
       location  label='Location',
       room      label='Rm. #',
       phone     label='Ext.'
from vsample.emp(pw=&mypw)
order by 1;

```

```

*****;
* View Name: vsample.payrank1 *;
* *;
* Protected: Yes *;
* *;
* Engine: SASesQL *;
* *;
* Uses: vsample.emp (view) *;
* *;
* Sorted By: descending salary *;
* *;
* Function: Performs frequency counts on all *;
* non-missing salary rates. Setup *;
* view #1 for pay rank analysis *;
* views. *;
* *;
*****;
create view vsample.payrank1(pw=&myPW) as
select salary, count(*) as count
from vsample.emp(pw=&myPW)
where salary is not missing
group by 1
order by 1 desc;

*****;
* *;
* View Name: vsample.salrank *;
* *;
* Protected: Yes *;
* *;
* Engine: SASesQL *;
* *;
* Uses: vsample.emp (view) *;
* vsample.payrank2 (view) *;
* *;
* Sorted By: salrank *;
* *;
* Function: Individual Employee Salary *;
* Ranking Information. *;
* *;
*****;
create view vsample.salrank(pw=&myPW) as
select a.empno, a.gender, a.salary,
       b.salrank format=6.2 label='Salary Rank',
       case
         when salrank<=0.25*max(salrank) then 1
         when 0.25*max(salrank)<salrank<=0.5*max(salrank) then 2
         when 0.5*max(salrank) <salrank<=0.75*max(salrank) then 3
         else 4
       end as quartile label='Quartile'
from vsample.emp(pw=&myPW) a,
     vsample.payrank2(pw=&myPW) b
where a.salary = b.salary
order by 4;

```

```

*****;
*
* View Name: vsample.rankstat
*
* Protected: No
*
* Engine: SASesQL
*
* Uses: vsample.salrank (view)
*
* Sorted By: gender
*
* Function: Salary Summary by Gender
*
*****;
create view vsample.rankstat as
select gender,
       count(*)          as count  label='Count',
       mean(salary)      as mpay   format=dollar10.2 label='Avg. Salary',
       std(salary)       as sdpay  format=dollar10.2 label='STD Salary',
       min(salary)       as lopay   format=dollar8.   label='Worst Salary',
       max(salary)       as hipay   format=dollar8.   label='Best Salary',
       mean(salrank)     as mpayr   format=6.2        label='Avg. Salary Rank',
       std(salrank)      as sdpayr  format=6.2        label='STD Salary Rank',
       stderr(salrank)   as sepayr  format=6.2        label='SE Salary Rank',
       max(salrank)      as lopayr  label='Worst Salary Rank',
       min(salrank)      as hipayr  label='Best Salary Rank'
from vsample.salrank (pw=&myppw)
group by 1;

*****;
*
* View Name: vsample.qtile1
*
* Protected: Yes
*
* Engine: SASesQL
*
* Uses: vsample.salrank (view)
*
* Sorted By: quartile gender
*
* Function: Performs frequency count on
*           salary rank quartiles by gender.
*           Setup view for salary ranks
*           quartile summary view.
*
*****;
create view vsample.qtile1 (pw=&myppw) as
select quartile, gender,
       count(*) as count label='Count'
from vsample.salrank (pw=&myppw)
group by quartile, gender;

```

```

*****;
*
* View Name: vsample.quartile
*
* Protected: No
*
* Engine: SAS/SQL
*
* Uses: vsample.qtile1 X 2 (view)
*
* Function: Generates single row summary view
*           on salary rank quartile counts by
*           gender.
*
*****;

```

```

create view vsample.quartile as
select a.quartile,
       a.count as mcount label='n Male',
       100*a.count/sum(a.count,b.count) as mpercent
       format=5.1 label='% Male',
       b.count as fcount label='n Female',
       100*b.count/sum(a.count,b.count) as fpercent
       format=5.1 label='% Female'
from vsample.qtile1 (pw=&myPW) a,
     vsample.qtile1 (pw=&myPW) b
where a.quartile = b.quartile and
      a.gender = 'M' and
      b.gender = 'F'
group by 1;

```

```

*****;
*
* View Name: vsample.paytest
*
* Protected: No
*
* Engine: SAS/SQL
*
* Uses: vsample.rankstat X 2 (view)
*
* Function: Generates single row summary view
*           with one-tailed p-value from
*           independant t-test of salary rank
*           using gender as the class.
*
*****;

```

```

create view vsample.paytest as
select a.count as mcount label='Number of Males',
       a.mpayr as mmpay label='Avg. Male Salary Rank',
       a.hipayr as mhipay label='Best Male Salary Rank',
       b.count as fcount label='Number of Females',
       b.mpayr as fmpay label='Avg. Female Salary Rank',
       b.hipayr as fhipay label='Best Female Salary Rank',
       case

```

```

when a.mpayr >= b.mpayr
  then 1-probt((a.mpayr-b.mpayr)/
    (sum(a.sepayr**2,b.sepayr**2)**0.5),
    sum(a.count,b.count,-2))
  else probt((a.mpayr-b.mpayr)/
    (sum(a.sepayr**2,b.sepayr**2)**0.5),
    sum(a.count,b.count,-2))
end as pvalue format=5.3
  label='p-Value Ho:<Avg.MaleRank>=<Avg.FemaleRank>'
from vsample.rankstat a,
  vsample.rankstat b
where a.gender = 'M' and
  b.gender = 'F';
quit;

* Create Data step Views *;
*****;
*
* View Name: vsample.payrank2
*
* Protected: Yes
*
* Engine: SASDSV
*
* Uses: vsample.payrank1 (view)
*
* Sorted By: descending salary
*
* Function: Calculates salary ranks with ties
*           from salary frequency counts.
*           Setup view #2 for salary ranks
*           analysis views.
*
*****;
data vsample.payrank2 (pw=&mypw
  keep=salary count salrank
  label='Pay Rank Setup View II')/
view=vsample.payrank2;
set vsample.payrank1 (pw=&mypw);
  by descending salary;

* Initialized Low and High Range *;
retain low 1 high 0;

* Use Frequency Count to Set Range *;
high = low + count - 1;

* Initialize Rank Sum to Zero *;
sumval = 0;

* Sum All Ranks Covered by Salary Count *;
do i = low to high;
  sumval = sumval + i;
end;

```

```

* Set Salary Rank to Average of      *;
* All Ranks Covered by Salary Count *;
salrank = sumval/count;

* Output Ranks with Ties *;
output;

* Set Low End of Range for Next *;
* Highest Salary for Repeat of *;
* Rank Calculation Process.      *;
low = high + 1;
run;

* Set OS Protection to Read-Only for All *;
* Datasets and Views.              *;
x 'chmod 444 /$HOME/viewdemo/vsample/*.*';
* End of Program *;

```

```

* SAS View Samples
*
* SAS Version: 6.11 for Solaris I
*
* Program: payraise.sas
*
* Function: Demonstrates how views are used like SAS datasets in
*           source code but provide the latest info from source
*           data without having to generate intermediate datasets.
*
* Programmer: Adam Hendricks, ahendric@icos.com, (206)485-1900 x2295
*
* Date: 5/27/96
*
* Update:
*
*;

* View Demo Lib *;
libname vsample '$HOME/viewdemo/vsample';

* First Run Through Views *;

* Salary Rank by Gender Summary *;
proc print data=vsample.rankstat label noobs;
  title 'Rank Statistics View Before Raise';
run;

* Quartile Counts of Salary Rank by Gender *;
proc print data=vsample.quartile label noobs;
  title 'Quartile View Before Raise';
run;

* t-Test of Salary Rank by Gender *;
proc print data=vsample.paytest label noobs;
  title 't-Test View Before Raise';
run;

* Give All Females a 2.5% Raise *;
x 'chmod 644 $HOME/viewdemo/vsample/*. *';
proc sql;
  update vsample.salary(pw=nopeek)
  set salary = salary*1.025
  where idnum in(select idnum
                  from vsample.employee(pw=nopeek)
                  where gender = 'F');
x 'chmod 444 $HOME/viewdemo/vsample/*. *';

* Second Run Through Views *;

* Salary Rank by Gender Summary *;
proc print data=vsample.rankstat label noobs;
  title 'Rank Statistics View After Raise';
run;

```

```

* Quartile Counts of Salary Rank by Gender *;
proc print data=vsample.quartile label noobs;
  title 'Quartile View After Raise';
run;

* t-Test of Salary Rank by Gender *;
proc print data=vsample.paytest label noobs;
  title 't-Test View After Raise';
run;

* Reset Salaries to Original Levels *;
x 'chmod 644 $HOME/viewdemo/vsample/*. *';
proc sql;
  update vsample.salary(pw=nopeek)
  set salary = salary/1.025
  where idnum in(select idnum
                  from vsample.employee(pw=nopeek)
                  where gender = 'F');
x 'chmod 444 $HOME/viewdemo/vsample/*. *';

* End of Programs *;

```

Rank Statistics View Before Raise

13:18 Wednesday, May 29, 1996 1

Gender	Count	Avg. Salary	STD Salary	Worst Salary	Best Salary	Avg. Salary Rank	STD Salary Rank	SE Salary Rank	Worst Salary Rank	Best Salary Rank
F	104	\$44,995.19	\$24,595.81	\$12,500	\$183,000	145.43	86.64	8.50	307	4
M	204	\$46,592.65	\$44,111.92	\$12,000	\$500,000	159.12	90.03	6.30	308	1

Quartile View Before Raise

13:18 Wednesday, May 29, 1996 2

Quartile	n Male	% Male	n Female	% Female
1	49	64.5	27	35.5
2	52	61.9	32	38.1
3	44	62.0	27	38.0
4	59	76.6	18	23.4

t-Test View Before Raise

13:18 Wednesday, May 29, 1996 3

Number of Males	Avg. Male Salary Rank	Best Male Salary Rank	Number of Females	Avg. Female Salary Rank	Best Female Salary Rank	p-Value Ho: <Avg.MaleRank>=<Avg.FemaleRan
204	159.12	1	104	145.43	4	0.098

Rank Statistics View After Raise

13:18 Wednesday, May 29, 1996 4

Gender	Count	Avg. Salary	STD Salary	Worst Salary	Best Salary	Avg. Salary Rank	STD Salary Rank	SE Salary Rank	Worst Salary Rank	Best Salary Rank
F	104	\$46,120.07	\$25,210.71	\$12,813	\$187,575	142.10	85.76	8.41	307	4
M	204	\$46,592.65	\$44,111.92	\$12,000	\$500,000	160.82	90.20	6.32	308	1

Quartile View After Raise

13:18 Wednesday, May 29, 1996 5

Quartile	n Male	% Male	n Female	% Female
1	49	63.6	28	36.4
2	52	62.7	31	37.3
3	44	62.0	27	38.0
4	59	76.6	18	23.4

t-Test View After Raise

13:18 Wednesday, May 29, 1996 6

Number of Males	Avg. Male Salary Rank	Best Male Salary Rank	Number of Females	Avg. Female Salary Rank	Best Female Salary Rank	p-Value Ho: <Avg.MaleRank>=<Avg.FemaleRan
204	160.82	1	104	142.10	4	0.038

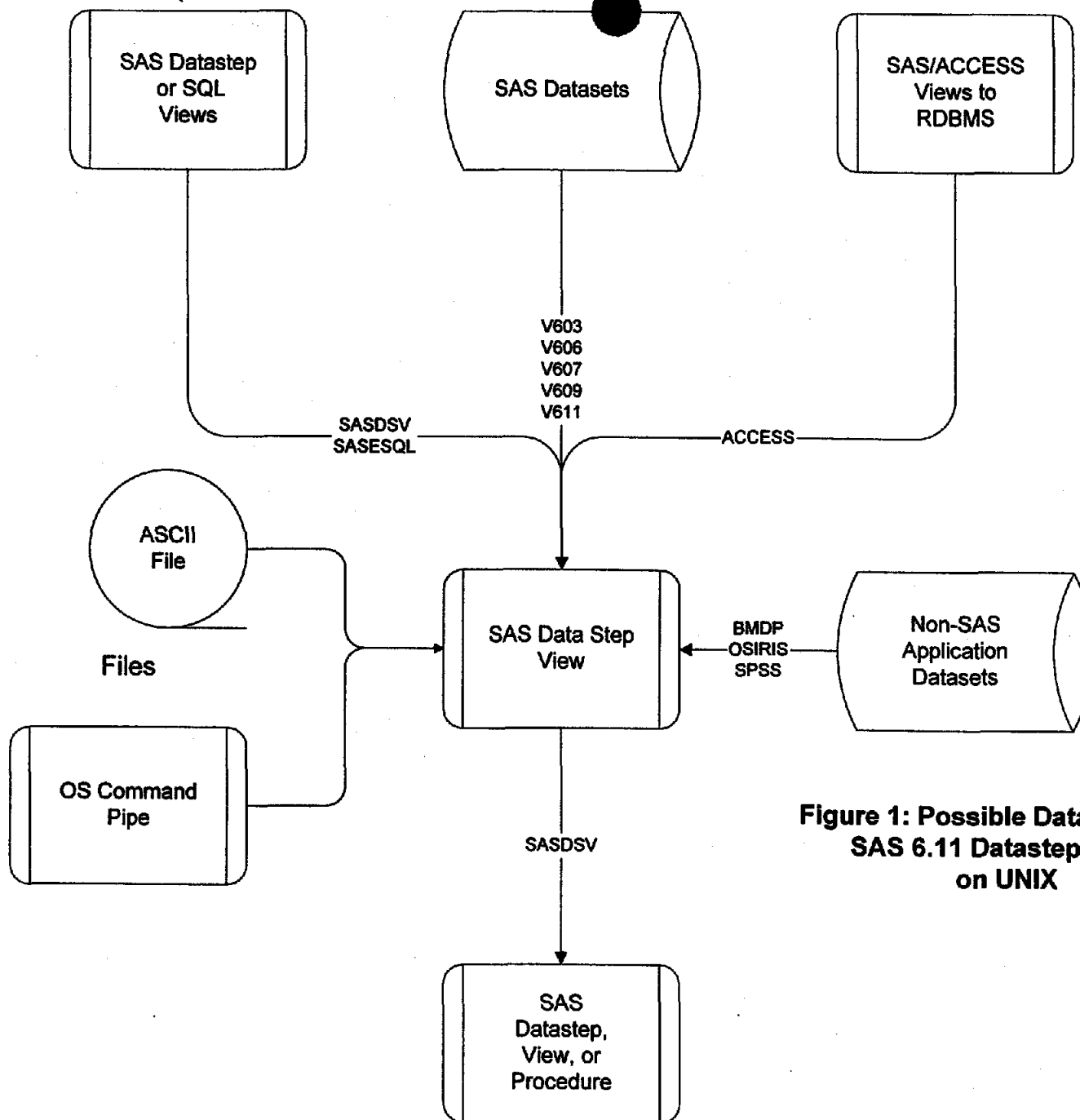


Figure 1: Possible Data Flows for SAS 6.11 Datastep Views on UNIX

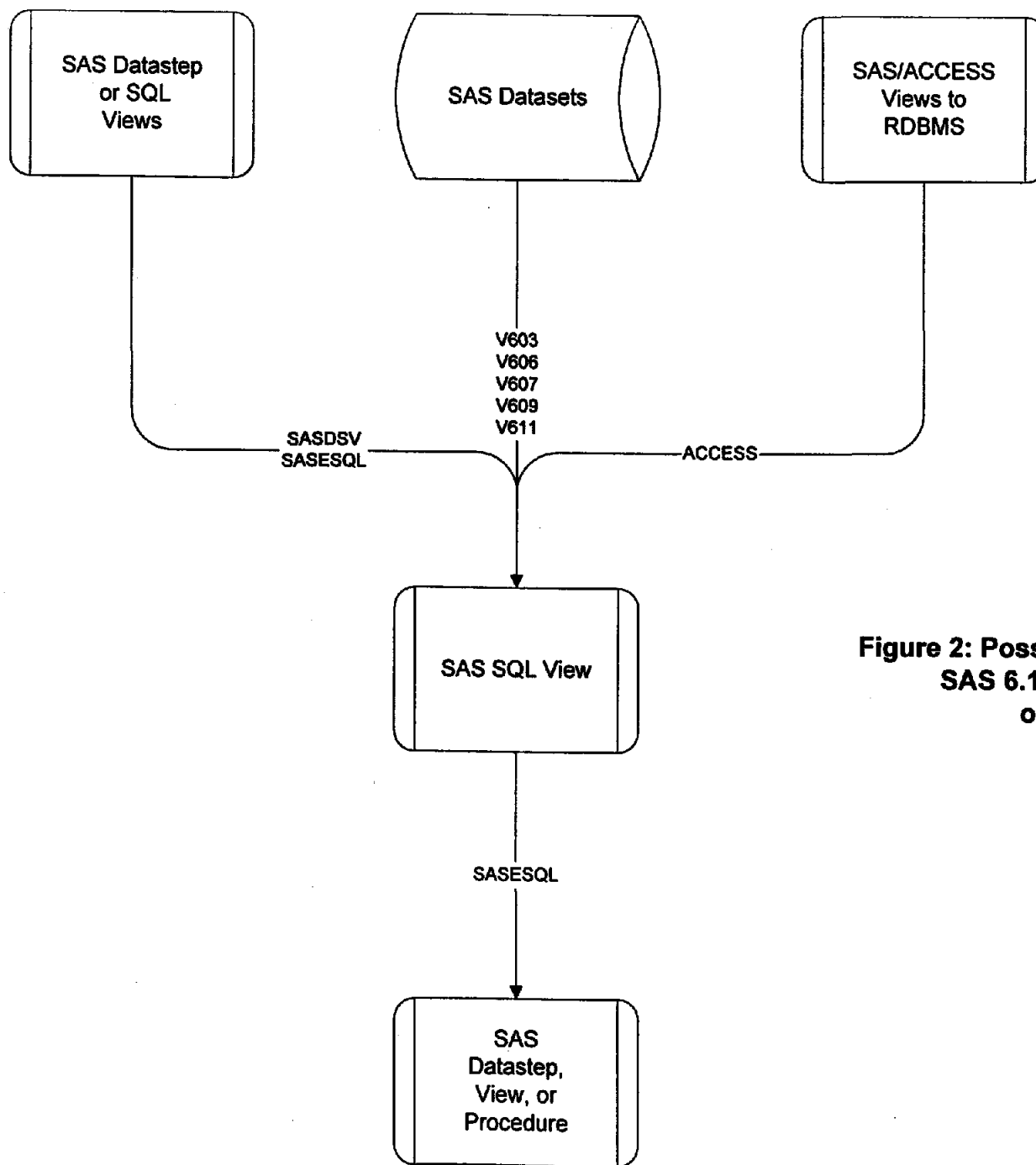


Figure 2: Possible Data Flows for SAS 6.11 SQL Views on UNIX

EMPLOYEE Password: NOPEEKIN

EMPNO	NAME	DEPTNO	JOB	CODE	SSNUMBER
01	BUNDY	2	05	098-43-0087	
02	CHEN	4	06	345-22-4456	
03	ROBERTS	3	09	123-45-6789	
04	RASPUTIN	1	03	666-66-6666	
05	SIEGAL	2	04	987-65-4321	
06	HUBILLA	3	08	333-82-1984	
07	LISTER	4	07	010-19-9983	
08	EINSTEIN	4	07	877-21-7777	
09	ABBOTT	1	01	999-99-9999	
10	COSTELLO	1	02	000-00-0000	

DEPT Password: MYOB

DEPTNO	DEPT	BUDGET
1	Administration	100000000
2	Marketing	500000000
3	Information Systems	30.78
4	Production	100000

JOBS Password: STAYOUT

JOB	CODE	TITLE	SALARY	BONUS
01	CEO		2000000	1,000,000 shares of stock
02	CFO		1000000	500,000 shares of stock
03	COO		1000000	500,000 shares of stock
04	VP Advertising		400000	100,000 shares of stock
05	Sr. VP Marketing		500000	200,000 shares of stock
06	Foreman		40000	A tee shirt
07	Assembler		15000	A bag of dirt
08	IS Project Manager		50000	A bunch of promises
09	Programmer		30000	A set of wrist splints

SAS Datasets

SAS Views

```
proc sql;
create view data.v_open as
select a.name, c.title, b.dept
from data.emp (pw=nopeekin) a,
data.dept(pw=myob) b,
data.jobs(pw=stayout) c
where a.deptno = b.deptno and
a.jobcode = c.jobcode
order by name;
```

```
proc sql;
create view data.v_secure(pw=abcdefgh) as
select a.name, c.title, b.dept,
c.salary format=dollar10., c.bonus
from data.emp (pw=nopeekin) a,
data.dept(pw=myob) b,
data.jobs(pw=stayout) c
where a.deptno = b.deptno and
a.jobcode = c.jobcode
order by salary desc, name;
```

Figure 3.

SAS View Output

V_OPEN

NAME	TITLE	DEPT
ABBOTT	CEO	Administration
BUNDY	Sr. VP Marketing	Marketing
CHEN	Foreman	Production
COSTELLO	CFO	Administration
EINSTEIN	Assembler	Production
HUBILLA	IS Project Manager	Information Systems
LISTER	Assembler	Production
RASPUTIN	COO	Administration
ROBERTS	Programmer	Information Systems
SIEGAL	VP Advertising	Marketing

V_SECURE Password: ABCDEFGH

NAME	TITLE	DEPT	SALARY	BONUS
ABBOTT	CEO	Administration	\$2,000,000	1,000,000 shares of stock
COSTELLO	CFO	Administration	\$1,000,000	500,000 shares of stock
RASPUTIN	COO	Administration	\$1,000,000	500,000 shares of stock
BUNDY	Sr. VP Marketing	Marketing	\$500,000	200,000 shares of stock
SIEGAL	VP Advertising	Marketing	\$400,000	100,000 shares of stock
HUBILLA	IS Project Manager	Information Systems	\$50,000	A bunch of promises
CHEN	Foreman	Production	\$40,000	A tee shirt
ROBERTS	Programmer	Information Systems	\$30,000	A set of wrist splints
EINSTEIN	Assembler	Production	\$15,000	A bag of dirt
LISTER	Assembler	Production	\$15,000	A bag of dirt

EMP Password: NOPEEKIN

EMPNO	NAME	DEPTNO	JOBCODE	SSNUMBER
01	BUNDY	2	05	098-43-0087
02	CHEN	4	06	345-22-4456
03	ROBERTS	3	09	123-45-6789
04	RASPUTIN	1	03	666-66-6666
05	SIEGAL	2	04	987-65-4321
06	HUBILLA	3	08	333-82-1984
07	LISTER	4	07	010-19-9983
08	EINSTEIN	4	07	877-21-7777
09	ABBOTT	1	01	999-99-9999
10	COSTELLO	1	02	000-00-0000

Administration Changed to
Supreme Rulers

DEPT Password: MYOB

DEPTNO	DEPT	OBUDGET
1	Supreme Rulers	100000000
2	Marketing	500000000
3	Information Systems	30.78
4	Production	100000

JOB5 Password: STAYOUT

JOBCODE	TITLE	SALARY	BONUS
01	CEO	2000000	1,000,000 shares of stock
02	CFO	1000000	500,000 shares of stock
03	COO	1000000	500,000 shares of stock
04	VP Advertising	400000	100,000 shares of stock
05	Sr. VP Marketing	500000	200,000 shares of stock
06	Foreman	40000	A tee shirt
07	Assembler	15000	A bag of dirt
08	IS Project Manager	50000	A bunch of promises
09	Programmer	30000	A set of wrist splints

SAS Datasets

SAS Views

```
proc sql;
create view data.v_open as
select a.name, c.title, b.dept
from data.emp (pw=nopeekin) a,
     data.dept(pw=myob) b,
     data.jobs(pw=stayout) c
where a.deptno = b.deptno and
     a.jobcode = c.jobcode
order by name;
```

```
proc sql;
create view data.v_secure(pw=abcdefgh) as
select a.name, c.title, b.dept,
     c.salary format=dollar10., c.bonus
from data.emp (pw=nopeekin) a,
     data.dept(pw=myob) b,
     data.jobs(pw=stayout) c
where a.deptno = b.deptno and
     a.jobcode = c.jobcode
order by salary desc, name;
```

Figure 4.

SAS View Output

V_OPEN

NAME	TITLE	DEPT
ABBOTT	CEO	Supreme Rulers
BUNDY	Sr. VP Marketing	Marketing
CHEN	Foreman	Production
COSTELLO	CFO	Supreme Rulers
EINSTEIN	Assembler	Production
HUBILLA	IS Project Manager	Information Systems
LISTER	Assembler	Production
RASPUTIN	COO	Supreme Rulers
ROBERTS	Programmer	Information Systems
SIEGAL	VP Advertising	Marketing

V_SECURE Password: ABCDEFGH

NAME	TITLE	DEPT	SALARY	BONUS
ABBOTT	CEO	Supreme Rulers	\$2,000,000	1,000,000 shares of stock
COSTELLO	CFO	Supreme Rulers	\$1,000,000	500,000 shares of stock
RASPUTIN	COO	Supreme Rulers	\$1,000,000	500,000 shares of stock
BUNDY	Sr. VP Marketing	Marketing	\$500,000	200,000 shares of stock
SIEGAL	VP Advertising	Marketing	\$400,000	100,000 shares of stock
HUBILLA	IS Project Manager	Information Systems	\$50,000	A bunch of promises
CHEN	Foreman	Production	\$40,000	A tee shirt
ROBERTS	Programmer	Information Systems	\$30,000	A set of wrist splints
EINSTEIN	Assembler	Production	\$15,000	A bag of dirt
LISTER	Assembler	Production	\$15,000	A bag of dirt

V_SECURE Password: ABCDEFGH

NAME	TITLE	DEPT	SALARY	BONUS
ABBOTT	CEO	Administration	\$2,000,000	1,000,000 shares of stock
COSTELLO	CFO	Administration	\$1,000,000	500,000 shares of stock
RASPUTIN	COO	Administration	\$1,000,000	500,000 shares of stock
BUNDY	Sr. VP Marketing	Marketing	\$500,000	200,000 shares of stock
SIEGAL	VP Advertising	Marketing	\$400,000	100,000 shares of stock
HUBILLA	IS Project Manager	Information Systems	\$50,000	A bunch of promises
CHEN	Foreman	Production	\$40,000	A tee shirt
ROBERTS	Programmer	Information Systems	\$30,000	A set of wrist splints
EINSTEIN	Assembler	Production	\$15,000	A bag of dirt
LISTER	Assembler	Production	\$15,000	A bag of dirt

Figure 5. Summary View

```
proc sql;
create view data.v_means as
select dept,
       mean(salary) as meansal
       format=dollar13.2
from data.v_secure(pw=abcdefgh)
group by dept
order by meansal desc;
```

DEPT	V_MEANS MEANSAL
Administration	\$1,333,333.33
Marketing	\$450,000.00
Information Systems	\$40,000.00
Production	\$23,333.33

SAS View
Output