ABSTRACT

This paper discusses how data set features such as compression, SQL views, deleted observations, and WHERE clause processing affect results from certain Screen Control Language (SCL) functions in Version 6 of SAS/AF® and SAS/FSP® software. The SCL functions discussed include LOCATEC, CUROBS, FETCHOBS, FETCH, ATTRN, NOTE, and POINT.

INTRODUCTION

With Version 6 of SAS/AF and SAS/FSP software, the flexibility of SCL provides many functions to manipulate data sets and views. Version 6 also adds more efficiency to data sets with data set features such as WHERE clause processing and compression of data sets. This paper discusses how certain data set features can affect processing of selected SCL functions.

SAS data set features discussed include the following:

- compressed data sets
- SQL views
- deleted observations
- WHERE clause processing.

All of these data set features affect observation numbering. This paper compares processing of certain SCL functions on a regular data set with data sets that have each feature listed. While there are two different types of views, such as SQL (Structured Query Language) and ACCESS views, this paper focuses on SQL views.

DATA SET FEATURES

Compressed SAS® Data Sets

Version 6 allows SAS data sets to be compressed. This compression reduces the amount of space required to store a SAS data set on disk. Space is reduced by collapsing consecutive repeating characters into fewer bytes. Therefore, each observation in a compressed data set may have a different length. Since each observation may have a different length and the SAS System cannot quickly calculate the location of each observation, access by observation number is no longer allowed after the data set is compressed.

The trade-off for the amount of space saved is more required CPU time. Compressed data sets require more CPU time to prepare records for I/O by expanding observations back to their original length.

SQL Views

The SQL procedure enables you to create logical tables or views of your data. In creating a view, you can retrieve data as they exist in a data set, subset the data with the WHERE clause processing, or combine data with join operations.

SCL can read a view as if the view were a SAS data file. Observation numbers displayed for the view reflect the order of the data in each view created and, therefore, do not typically correspond to observation numbers for the same data in a SAS data file. A SAS data set in Version 6 includes both members of type DATA and members of type VIEW.

Deleted Observations

In Version 5 of the FSEDIT procedure, the DELETE command simply sets an observation's values to missing. The observation is still accessible, but the values are gone. To remove deleted observations from the data set, you need to execute a DATA step.

With the Version 6.06 engine, the DELETE command marks the observation for deletion from the data set and no further editing is allowed. Once you move to another observation, you cannot return to a deleted observation.

While the deleted observation is no longer accessible, observation numbers are not renumbered automatically. This enables you to continue to use observation numbers as pointers. For example, if you delete observation 3, the PRINT procedure would display nothing for observation 3. It would actually skip the number 3 when listing observations and show information for observations 1, 2, and 4.

Observations are not renumbered intentionally to

- allow use of observation numbers as pointers. This is important when using PROC FSEDIT or subsequent SAS code that directly accesses observations by number, such as with the POINT= option in a SET statement.
- allow the applications programmer to control observation renumbering if necessary.

There are several ways to renumber observations. These include the following:

- sort the data set
- set the data set using the SET statement
- copy the data set with the COPY procedure.

WHERE Clause Processing

In Screen Control Language, the WHERE function establishes a set of criteria for reading SAS data set observations by applying a WHERE clause on a data set. A WHERE clause is a set of conditions that observations in a data set must meet in order to be processed. Any WHERE clause applied by the SCL WHERE function is temporary and can be modified or canceled by subsequent SCL statements.

WHERE clause processing enables you to subset a SAS data set based on certain criteria. Once the WHERE clause is in effect, direct access by observation number in PROC FSEDIT is no longer allowed.

One feature of WHERE clause processing is that it takes advantage of indexed data sets to process data and retrieve values much quicker.

These are just some of the new features for SAS data sets in Version 6. As you can tell, compression, SQL views, deleted observations, and WHERE clause processing each affect SAS data sets differently. In the same manner, they impose their own restrictions.
on SCL functions that manipulate data. The following examples compare results from selected SCL functions with a regular data set and one with each of these data set features.

**EXAMPLES**

The following examples compare the effects of compressed data sets, SQL views, deleted observations, and WHERE clause processing on selected SCL functions.

The original data set (CLASS.ORIGINAL) contains three variables: NAME, SEX, and AGE. This data set contains five females and five males and is sorted by NAME. The compressed data set (CLASS.COMPRESS) is the compressed version of the original data set. In creating the SQL view (CLASS.VIEW), we included each observation from the original data set in the same order. To show deleted observations in CLASS.DELETE, we deleted observation 3 whose value for the variable NAME is CARTER. We used the WHERE function in SCL to apply a WHERE clause to CLASS.WHERE that subsets the data set by females.

The following code is used to create the original data set:

```sas
data class.original;
input name $ sex $ age;
cards;
  ANDREWS M 15
  BAKER F 15
  CARTER F 111
  GOODMAN F 15
  HARRIS M 15
  MARTIN M 16
  MELLEN F 15
  NANCE M 15
  SELLERS M 15
  TAYLOR F 15
run;
```

A PROC PRINT of CLASS.ORIGINAL appears in Output 1.1.

<table>
<thead>
<tr>
<th></th>
<th>NAME</th>
<th>SEX</th>
<th>AGE</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>ANDREWS</td>
<td>M</td>
<td>15</td>
</tr>
<tr>
<td>2</td>
<td>BAKER</td>
<td>F</td>
<td>15</td>
</tr>
<tr>
<td>3</td>
<td>CARTER</td>
<td>F</td>
<td>111</td>
</tr>
<tr>
<td>4</td>
<td>GOODMAN</td>
<td>F</td>
<td>15</td>
</tr>
<tr>
<td>5</td>
<td>HARRIS</td>
<td>M</td>
<td>15</td>
</tr>
<tr>
<td>6</td>
<td>MARTIN</td>
<td>M</td>
<td>16</td>
</tr>
<tr>
<td>7</td>
<td>MELLEN</td>
<td>F</td>
<td>15</td>
</tr>
<tr>
<td>8</td>
<td>NANCE</td>
<td>M</td>
<td>15</td>
</tr>
<tr>
<td>9</td>
<td>SELLERS</td>
<td>M</td>
<td>15</td>
</tr>
<tr>
<td>10</td>
<td>TAYLOR</td>
<td>F</td>
<td>15</td>
</tr>
</tbody>
</table>

The following code is used to create the compressed data set:

```sas
data class.compress(compress=yes);
set class.original;
run;
```

The following code is used to create the SQL view selecting all variables from the original data set:

```sas
proc sql;
create view class.view as
  select * from class.original;
quit;
```

For the data set with a deleted observation, we set the original data set and then used the DELETE command in PROC FSEDIT to delete observation 3.

```sas
data class.delete;
set class.original;
proc fsedit data=class.delete;
run;
```

To apply WHERE clause processing, we also set the original data set and then used the WHERE function in SCL to apply the WHERE clause.

```sas
data class.where;
set class.original;
run;
```

The SCL code includes:

```sas
dsidopen('class.where','i');
rwhere(dsid,'sex = "F"');
```

**SCL Functions That Deal With Data Set Observations**

The applications programmer needs to be aware of the effects of compressed data sets, views, deleted observations, and WHERE clause processing on the results of SCL functions. In the following sections, we compare results from the following SCL functions using data with each of these features:

- LOCATEC
- CUROBS
- FETCHOBS
- FETCH
- ATTRN

**LOCATEC (LOCATEN)**

The LOCATEC function searches a data set for an observation that contains a specified value. This function returns a number greater than 0 to indicate that the value was found. Since LOCATEC skips deleted observations and counts only nondeleted observations, the value returned may not correspond to the observation number where the specified value was found.

Refer to the data for CLASS.ORIGINAL and notice that GOODMAN is the fourth value listed for the variable NAME. Compare the results of the LOCATEC function for each data set feature when a sequential search is performed and the search value specified is GOODMAN. The LOCATEC function is specified as follows:

```sas
val=locatec(dsid, varnum(dsid, 'name'), 'GOODMAN', 'u');
```

<table>
<thead>
<tr>
<th>DS</th>
<th>VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>CLASS.ORIGINAL</td>
<td>4</td>
</tr>
<tr>
<td>CLASS.COMPRESS</td>
<td>4</td>
</tr>
<tr>
<td>CLASS.VIEW</td>
<td>4</td>
</tr>
<tr>
<td>CLASS.DELETE</td>
<td>3</td>
</tr>
<tr>
<td>CLASS.WHERE</td>
<td>4</td>
</tr>
</tbody>
</table>

Even though GOODMAN was found in the fourth observation, the data set with a deleted observation, CLASS.DELETE, returns the value 3. This is because the deleted observation was not counted by the LOCATEC function. Since different data set features can cause LOCATEC to return varying numbers, the important point to note is that we know the search value was found since the value returned from LOCATEC was greater than 0.

Another point to note is that GOODMAN is a female so this observation meets the WHERE condition. If the value for which we were
searching had not met the WHERE condition, LOCATEC would have returned a 0 to indicate the value was not found. With a WHERE clause active, LOCATEC only searches the observations that meet the WHERE condition and then returns the actual observation number where the value was found.

Now notice the impact each feature has on the results when the LOCATE function is used to perform a binary search specified as follows:

\[
\begin{align*}
\text{val} &= \text{locatec}(\text{dsid, varnum(\text{dsid, 'name'})}, 'GOODMAN', 'u');\\
\text{VAL} &= 4\\
\text{CLASS.ORIGINAL} &= 4\\
\text{CLASS.COMPRESS} &= 0\\
\text{CLASS.VIEW} &= 0\\
\text{CLASS.DELETE} &= 0\\
\text{CLASS.WHERE} &= 0
\end{align*}
\]

Since none of these features allow sequential numbering of observations, LOCATEC cannot perform a binary search and, therefore, returns the value 0. Include the following code to check the value for SYMSMG to see why the search could not be performed.

\[
\_msg = \text{stderr}();
\]

For CLASS.DELETE the following message displays on the message line:

Cannot do binary search on dataset with deleted observations.

CUROBS

The CUROBS function returns the actual observation number of the observation that has been fetched into memory. Unlike the LOCATEC function, the value returned for CUROBS does include deleted observations. Compare the previous results for a sequential search with LOCATEC to the value returned for CUROBS specified as follows:

\[
\begin{align*}
\text{rc} &= \text{fetch}(\text{dsid});\\
\text{name} &= \text{getvarc}(\text{dsid, varnum(\text{dsid, 'name'})});\\
\text{VAL} &= \text{curobss}(\text{dsid});\\
\text{VAL} &= 4\\
\text{CLASS.ORIGINAL} &= 4\\
\text{CLASS.COMPRESS} &= 4\\
\text{CLASS.VIEW} &= 4\\
\text{CLASS.DELETE} &= 4\\
\text{CLASS.WHERE} &= 4
\end{align*}
\]

Notice that while LOCATEC returned the value 3 for CLASS.DELETE, CUROBS returns 4 as the actual observation number where GOODMAN was found. This difference in the value returned for LOCATEC and CUROBS should alert the applications programmer that the data set contains deleted observations. Since the compressed data set does not allow access by observation number, the value it returns to CUROBS is missing.

Since CLASS.VIEW was created with the same observation values as in CLASS.ORIGINAL, it also has GOODMAN in the fourth observation and, therefore, returns the value 4 for CUROBS. In the case of a view, the value returned by CUROBS indicates the order of the observations in the table created.

FETCHOBS

The FETCHOBS function fetches the specified observation from a data set into memory. FETCHOBS treats an observation number as a relative number and counts only observations that meet any restrictions imposed. For example, when a WHERE clause is active, FETCHOBS only counts observations that meet the WHERE condition.

Refer to CLASS.ORIGINAL and compare the results from FETCHOBS when the fourth observation is specified. The syntax for FETCHOBS is as follows:

\[
\begin{align*}
\text{rc} &= \text{fetchobs}(\text{dsid, 4});\\
\text{name} &= \text{getvarc}(\text{dsid, varnum(\text{dsid, 'name'})});\\
\text{NAME} &= \text{GOODMAN}
\end{align*}
\]

Since the third observation is deleted from CLASS.DELETE, FETCHOBS does not count this observation in its search. Therefore, the fourth observation it reaches contains the value MILLER.

Also note that since a WHERE clause that selects only female class members is active for CLASS.WHERE, the fourth observation of this subset corresponds to the value MILLER.

FETCH

The FETCH function fetches the next nondeleted observation from the specified data set into memory. Like LOCATEC and FETCHOBS, FETCH also skips deleted observations.

Refer to CLASS.ORIGINAL and compare the value returned for NAME when the previous value fetched for NAME was BAKER and the following statements are used to fetch the next observation:

\[
\begin{align*}
\text{val} &= \text{locatec}(\text{dsid, varnum(\text{dsid, 'name'})}, 'BAKER', 'u');\\
\text{rc} &= \text{fetch}(\text{dsid});\\
\text{name} &= \text{getvarc}(\text{dsid, varnum(\text{dsid, 'name'})});\\
\text{NAME} &= \text{CARter}
\end{align*}
\]

Notice that GOODMAN is the value returned for NAME in CLASS.ORIGINAL since observation 3 (CARter) is deleted.

Since data set features such as compression, SQL views, deleted observations, and WHERE clause processing can affect values returned for certain SCL functions, it is important to know that these features apply to the data you are processing. The applications programmer can use the ATTRN function to determine the existence of these data set features and proceed accordingly.

ATTRN (ATTRC)

The ATTRN function returns the value of a numeric attribute for the specified data set. The values returned from this function help the applications programmer determine such information as whether or not the data set contains deleted observations, if a WHERE clause is active, and if access by observation number is allowed. To determine this information, specify the following attributes on the ATTRN function:

\[
\begin{align*}
\text{NDEL} &\quad \text{indicates the number of deleted observations in the data set} \\
\text{WHSTMT} &\quad \text{indicates if a WHERE clause is active and if so, which type} \\
\text{RADIX} &\quad \text{indicates if access by observation number is allowed}
\end{align*}
\]

Compare the effects of each data set feature on the NDEL attribute specified in SCL as follows:

1532
val=attrn(dsid,'ndel');

Since CLASS.DELETE is the only data set with a deleted observation, it is the only one that returned a value for ATTRN that is greater than 0. The value 1 indicates that CLASS.DELETE contains only one deleted observation.

Compare the effect of each data set feature on the WHSTMT attribute specified as follows:

\[
\text{val=attrn(dsid,'whstmt');}
\]

The value 1 returned for the RADIX attribute of CLASS.COMPRESS indicates that the compressed data set is the only one in our example that does not allow access by observation number.

SCL Functions That Do Not Rely on Observation Numbers

We have seen how compressed data sets, SQL views, deleted observations, and WHERE clause processing can affect values returned from certain SCL functions. With the addition of a few new Version 6 SCL functions, you can move through a secondary data set without referring to observation number at all.

These functions include:

- NOTE
- POINT

NOTE

The NOTE function returns an observation id for the current observation in a SAS data set. This id is then used to return to this observation with the POINT function. Example syntax for this statement is as follows:

\[
\text{noteid=note(dsid);}
\]

Example of an SCL Application

The following is an SCL application using the FSEDIT procedure that manipulates a data set without ever specifying an observation by number. By using NOTE, POINT, and FETCH functions, we can move through this data set without concern of the effects of data set features on our results. In this example, the user specifies a dollar range of sales by which to subset his secondary data set, as shown in Screen 1.

Screen 1 Initial FSEDIT Screen for Request

The Screen Control Language program uses a WHERE clause to subset the sales database on the values specified for two variables. SCL displays the first observation of the subset with a message that indicates how many observations met the search criteria. The user is instructed to press F2 to scroll the observations that met the search, as shown in Screen 2.

Screen 2 FSEDIT Screen Displaying Number Meeting Your Request

Each time the user scrolls to the next observation, a message indicates how many more observations remain to be viewed, as shown in Screen 3.
Screen 3  
FSEDIT Screen As You Scroll Each Observation

The SCL code for this example follows. Notice that the WHERE function is used to subset the data. The NOTE function is used to indicate the first observation of the subset. The POINT function returns to the first observation once the number of observations meeting the search criteria has been counted. The FETCH function fetches each observation within the subset. The SCL code enables you to move among observations of a data set without regard to the number of the observation being processed.

**FSEINIT:**

dsid=open('test.sales','i');
call set(dsid);
call setfkey(fkeyname(2),'CHECK'); /* set function key for scrolling */
return;

**INIT:**

control enter;
return;

**MAIN:**

if field('modified','lowval highval') > 0 then
   do;
      rctwhere(dsid,'sales GE ' || lowval, 'and sales LE ' || highval);
      /* Check to see if any observations met the WHERE clause. */
   end;
   /* Check to see if any observations met the WHERE clause. */
   num=attrn(dsid, 'any');
   if num = 0 then
      do;
         msg = 'No observations meet your request. Select again.';
         return;
      end;
   val=0;
   do while (fetch(dsid) ne -1); /* count observations that meet request*/
      val + 1;
      if val = 1 then noteid = note(dsid);
      rctpoint(dsid, noteid); /* return to 1st matching observation*/
      rctfetch(dsid);
      msg = val 1 ' obs meet your request. Press F2 to scroll these.';
      end;
      if word(1) = 'CHECK' then
         do;
            call nextcmd();
            rctfetch(dsid);
            if rc = -1 then msg = 'NO more obs meet your request. Select again.';
            else do;
               val = val - 1;
               msg = val 1 ' more obs meet your request';
            end;
         end;
      return;
   end;
   if word(1) = 'CHECK' then
      do;
         call nextcmd();
         rctfetch(dsid);
         if rc = -1 then msg = 'NO more obs meet your request. Select again.';
         else do;
            val = val - 1;
            msg = val 1 ' more obs meet your request';
         end;
      end;
      return;
   return;
   FSETERK:
      do;
         val = val - .
         msg = val 1...
      end;
      call close(dsid);
      return;

**CONCLUSION**

This paper discusses the effects of such data set features as compression, SQL views, deleted observations, and WHERE clause processing on certain SCL functions that manipulate data. While these features can be beneficial for data sets, the SCL applications programmer should be aware of their existence and their effect on function results.

SAS, SAS/AF, and SAS/FSP are registered trademarks of SAS Institute Inc., Cary, NC, USA.