



CCAPRINT

A Newsletter Excerpt for System 1032 Users

June 2006

USE OF AND ACCESS TO PRODUCTS AND FEATURES ARE IN ACCORDANCE WITH THE TERMS AND CONDITIONS OF THE USER'S SOFTWARE LICENSE. THE PRESENTATION OF MATERIAL HEREIN DOES NOT, IN ANY MANNER, MODIFY SUCH TERMS AND CONDITIONS.

Validating Key Table Values

By Tym Stegner

When you are about to update a key, you need to prove that the key table is valid at the outset. How *does* one verify that the values in an attribute's key table are correct and up to date? This article recommends methods of validation to inspect whether key table values are valid.

Re-key the Attribute(s)

When using the V9.82-2 version of System 1032 or later, keying an attribute always results in a completely valid key table. If you are uncertain of an attribute key table, the simple, safe remedy is to re-key the suspect attribute or the entire dataset.

Using VALUES Output for a Quick Inspection – Tip 1

The System 1032 VALUES command displays a values breakdown list for a specified attribute. The data values from the low-level key tables are scanned, and the value occurrences are counted and displayed. As the low-level tables must be fully scanned to perform the operation, any inconsistencies in the tables are noted by System 1032 and reported.

The drawback to this tip is that only the key table values are displayed. To get a more complete check of the accuracy of the values and their counts, you want to compare the actual data in the attribute records against the reported values from the low-level key table. Such processing is *not* done by the VALUES command.

Comparing VALUES Output with PRINT or DUMP Output

The easiest way to make this check is to compare the output of the VALUES command for an attribute to a summarized list of the attribute's values that was created by a PRINT (or a DUMP) command. (Do not use the PRINT BY command to create a summarized report, because PRINT BY optimizes its processing by using the attribute's key tables, if available, to report the values.)

I use the following approach to create a breakdown list of the values of an attribute in a dataset.

Extracting Values to a Flat File

First, issue a PRINT (or DUMP) command to extract all the attribute values out to a flat file. If you issue a PRINT command, be sure to include the WITHOUT TITLES clause to suppress the attribute titles. When using a DUMP command, I use text formats to facilitate the next steps: sorting and aggregation.

I recommend that when sorting the values *do not* sort the data in System 1032 unless you have a small dataset, because the I/O load of extracting the data in sorted order is higher than using the VMS Sort utility to sort the data in the flat file. You can just specify the command as:

```
$ SORT inputfile outputfile
```

However, it is more efficient to specify the /key quality to identify and quantify the sort information. See VMS HELP on sort/key for an explanation.

Extracting Unique Values and Counts – Tip 2

Now, we have a sorted list of values. Next we need a list of unique values, including a count. For this, I use variations on a simple DCL command file, shown in Figure 1, to aggregate the multiple values down to a single value.

Figure 1. Aggregating unique values

```
$! SQSHR.COM
$  Open/Write OcX 'Ff$parse(P1,,,"Name")'.sum
$  Open Icx 'P1                      !P1 is input file name.type
$  Read Icx Lval
$  Xval = ""
$  Lcn = 1
$Vloop:
$  Read/End=No_More Icx Xval
$  If Xval .Nes. Lval
$  Then
$    Write OcX F$fao("!AS|!SL",Lval,Lcn)
$    Lval = Xval
$    Lcn = 1
$  Else
$    Lcn = Lcn + 1
$  Endif
$  Goto Vloop
$No_More:
$  Write OcX F$fao("!AS|!SL",Lval,Lcn)
$  Close Icx
$  Close OcX
$ Exit
```

Using the Command File

The command file in Figure 1 opens a supplied input file, and creates an output file with the same file name, but with the new file type .SUM. A loop reads values one at a time,

writing a line to the output file when the value changes. At the end of the loop, a final write takes care of the last value. The output line is *value|count*.

For most datasets, you can compare the VALUES key-table count and the PRINT (or DUMP) record generated count for a match by visual inspection. For larger value sets, you might make use of the VMS differences/parallel command to more easily compare the two files. While the format differences create obvious incorrect matching, there is the side benefit of automatically presenting the files side-by-side, to facilitate visual inspection.

Extraction Unique Values and Counts – Tip 3

Figure 2 illustrates another simple DCL command file you can use to do a line-by-line comparison.

Figure 2. Creating a line-by-line comparison

```
$ Open Vc Valmode.Sum      !xx      (##)
$ Open Pc Prtmode.Sum      !xx|##
$Vloop:
$ Read/End=Nomore Vc Vrec
$ Read/End=Nomore Pc Prec
$ If F$edit(F$element(0,"|",Prec),"TRIM") .Nes. -
    F$edit(F$element(0,"(",Vrec),"TRIM")
$ Then
$   Write Sys$output "Out of sync"
$   Goto Nomore
$ Endif
$ If (F$element(1,"(",Vrec)-")") .Ne. F$element(1,"|",Prec)
$ Then
$   Write Sys$output "Mismatch:  ''Vrec' Vs. ''Prec'"
$ Endif
$ Goto Vloop
$Nomore:
$ Close Vc
$ Close Pc
$ Exit
```

Note: The code in Figure 2 excludes code necessary to handle the differing presentation of MISSING between DUMP or PRINT reports and the VALUES command.

Using a Find/Search Value Comparison – Tip 4

The code in Figures 1 and 2 validate the key tables only at the lowest levels, where the records and value information are stored. This code does not verify the higher-level key tables, used to simplify the query for specific values during FIND command processing.

To verify the higher-level tables and simultaneously check the low-level tables for complete value inclusion, we will use one of the previous files as the input to a more complete test of the key table integrity. The code in Figure 3 is specific to a particular attribute in a particular dataset, but you could write a tool to generate a similar procedure for any given keyed attribute in a dataset.

Using the summarized data file generated by PRINT or DUMP processing, the following procedure opens two instances of the parent dataset, and performs FIND and SEARCH commands against the steadily diminishing selection set to assure that both FIND and SEARCH processing locate the proper number of records from the input file. Lastly, it checks for any FIND or SEARCH records remaining after the queries.

Figure 3. Analyzing a specific attribute

```

! Checks MODE attribute in DETAIL table using PMODE.SUM input file

Open Ds DETAIL As FDS Readonly,-
    Ds DETAIL As SDS Readonly

Var Xrec,Xval Text Varying
Var Frn,Srn,Xrn Integer
Var Ctx Integer Init 0 !File context

Init 8 Mode.Err
Call Open_File(Ctx,"Prtmode.Sum",)
Set Message Info No Print

Set Ds SDS; Find All; Consider On
Set Ds FDS; Find All; Consider On

Repeat
    Call Read_File(Xrec,Ctx)
    If Xrec Ne Missing Then
        Let Xval = Xrec[1:$Find("|",Xrec)],
            Xrn = $Int(Xrec[($Find("|",Xrec)+1):$Len(Xrec)])

        Set Ds FDS
        Find Mode Xval
        Let Frn = $Nrec
        Find Not Last; Consider Replace

        Set Ds SDS
        Search Mode Eq Xval
        Let Srn = $Nrec
        Find Not Last; Consider Replace

        If Xrn Ne Frn OR Xrn Ne Srn Then
            Write On 8 Xval Xrn Frn Srn
            Write Xval Xrn Frn Srn
        End_If
    End_If

Until (Xrec Eq Missing)

Set Ds FDS; Find All
If $Nrec Ne 0 Then
    Write On 8 $Nrec Fmt( "****" 2x i5 " Leftover records after FIND")
    Write $Nrec Fmt( "****" 2x i5 " Leftover records after FIND")
    Print On 8 Mode
End_If

Set Ds SDS; Find All

```

```
If $Nrec Ne 0 Then
  Write On 8 $Nrec Fmt("****" 2x i5 " Leftover records after SEARCH")
  Write $Nrec Fmt("****" 2x i5 " Leftover records after SEARCH")
  Print On 8 Mode
End_If

Call Close_File(Ctx)

Exit
```

Notating the Code in Figure 3

1. Consider sets are used to cause the next FIND or SEARCH command to disregard already-located records. In each value cycle, the consider set is reduced to unqueried records.
2. An error file is produced that records any instances where FIND or SEARCH processing located a number of records different from the value read from the summary file.
3. The code at the bottom of Figure 3 checks to see if records remain after the summary files values are queried.

Figure 4 is a sample run using the PRTMODE.SUM file from Figure 3.

Figure 4. Using the PRTMODE SUM file

```
$ S1032 Use Check_Mode
Current Dataset Is Now FDS
Initializing Channel 8
%RMS-W-EOF, End Of File Detected
***      45 Leftover Records After FIND
***      45 Leftover Records After SEARCH
```

These results are as expected. My PRTMODE.SUM file had the missing value line removed. The 45-record discrepancy determined by the CHECK_MODE command file represents the records where MODE is MISSING.

In Summary

The previous tips are useful for checking to see if a key table in a dataset is inconsistent with itself or inconsistent with the data values in the dataset. Employing these tips does not eliminate the need for regular dataset maintenance, but can serve as a cautionary check on an already opened dataset, if necessary.

Re-keying an attribute necessarily requires exclusive access to the dataset, so these processes might be used to check for key table validity without shutting down an active subsystem.