INTRODUCTION

Some of the most useful and least heralded additions to Version 6.07 of the SAS System are dictionary tables and views. They contain a wealth of information about SAS datasets, catalogs, system options, and external files. Some of the information was available in earlier versions of the SAS System via cumbersome procedure output datasets. Other information was simply not accessible. The dictionary tables and views (which we will simply refer to as the "tables") put this information at the disposal of the application developer in a straightforward manner. Best of all, the tables are automatically created and maintained by the SAS System, suggesting that there really may be such a thing as a free lunch (or "low cost lunch," as we note in the Performance section).

This paper explains the organization of the tables, describes their content, presents suggestions for their use, and discusses several short applications of interest to SAS programmers. The reader should come away from the discussion with an understanding of the benefits of the dictionary tables as well as some ideas for how they may be used in their programming environments.

BACKGROUND

SAS programmers have always needed high-level information about the contents of SAS datasets and other entities. Consider a few scenarios.

- You want to determine if a SAS dataset exists. Possible solutions include processing a CONTENTS output dataset or a _NULL_ data step with strategically placed SET and STOP statements and calls to SYMPUT. These solutions are unnecessarily awkward.
- A project uses numerous SAS datasets, some of them with like-named variables. You want to identify instances of like-named variables with conflicting data types or lengths. For example, the variable GENDER may be stored as a character, length 1 in one dataset and numeric, length 3 in another. These data "scrubbing" activities are usually part of the Data Warehousing development cycle.
- You are using many macro variables in a program. During the course of development you become, for lack of a better word, hopelessly confused. You want to list all macro variable names and their values.
- A macro has to adjust various system options. You'd like to capture the current settings, reset the options, and when exiting the macro reset the options to their original values. This is, for all practical purposes, not possible in Base SAS software.

Dictionary tables make these tasks and others feasible with a minimal amount of coding effort. A brief outline of some of the tables' characteristics follows. They are:

- Only in Version 6.07 and higher.
- Automatically created during SAS System startup.
- Maintained automatically during the course of the interactive session or batch job. One caveat: the tables are tied to implicit (WORK-level) and explicit LIBNAMEs — files allocated in MVS systems with DD statements will not be found in the tables.
- Read-only: You cannot change the table or view organization. You can affect their contents by making changes in the SAS environment. Whenever you change a system option, create a dataset, delete a member from a catalog, add a label to a variable, etc. one or more tables are updated.
- Accessible from SQL with the reserved LIBNAME of DICTIONARY (yes, that's a 10-letter LIBNAME!)
- Accessible outside SQL by using views defined in the LIBNAME of SASHELP. SASHELP is allocated automatically during startup.
- Present and maintained during your session whether you use them or not. There is no system option to suppress them.

So much for what they are. Let's take a look at what's inside them.

TABLE AND VIEW ORGANIZATION

The SAS System currently maintains 10 dictionary tables (available in SQL) and 15 views derived from these tables (available with the reserved LIBNAME of SASHELP). The tables and their relationship to the views are presented in Exhibit 1. Note that with the exception of the MEMBERS, which is the basis for seven views, each table has a corresponding view in SASHELP. The breakout of the MEMBERS table is discussed in more detail later in this section.

Exhibit 1: Relationship Between Dictionary Tables and SASHELP Views

<table>
<thead>
<tr>
<th>Dictionary Table</th>
<th>SASHELP View(s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>catalogs</td>
<td>vssectlg</td>
</tr>
<tr>
<td>columns</td>
<td>vvolum</td>
</tr>
<tr>
<td>extfiles</td>
<td>vexfil</td>
</tr>
<tr>
<td>indexes</td>
<td>vxindex</td>
</tr>
<tr>
<td>macros</td>
<td>vxmacro</td>
</tr>
<tr>
<td>members</td>
<td>vsvlib, vstetable, vssectlg, vsaccess, vsview, vstabvw</td>
</tr>
<tr>
<td>options</td>
<td>vsoption</td>
</tr>
<tr>
<td>tables</td>
<td>vstable</td>
</tr>
<tr>
<td>titles</td>
<td>vstitle</td>
</tr>
<tr>
<td>views</td>
<td>vsview</td>
</tr>
</tbody>
</table>

*Available only in Version 6.11*

The rest of this section discusses each table and view. The output for the exhibits was generated by the following code:

```proc sql;
   describe table dictionary_table;
```

In the above, table is the name of the dictionary table of interest (COLUMNS, CATALOGS, etc.).

Several methods are available for examining view contents. The CONTENTS procedure could be used as follows:

```proc contents data=sashelp.view;
```

In the above, view is the name of the view of interest (volumn, vsectlg, etc.).

Another method of examining views is using the LIBNAME and DIR windows available in Display Manager. Exhibit 3 displays a DIR window, called directly from the Command Line of Display Manager or from within the LIBNAME window. The user can enter "S" next to the VIEW of interest to get field information similar to that given by the CONTENTS procedure. Entering "B" next to the view invokes the FSBROWSE procedure for the view (assuming, of course, that SAS/FSP software is installed at the user's site). Once the "B" command executes successfully the user sees a screen resembling that in Exhibit 3.

The full FSBROWSE command set is available for use when inspecting the views. A particularly useful feature is the ability to filter...
Advanced Tutorials

observations with a WHERE command. To find observations for a specific dataset, for example, enter:

```
WHERE libname='PERM' & memname='FISCAL95'
```

EXHIBIT 2: Using DIR and FSBROWSE to Examine a SASHELP View

EXHIBIT 3: Using FSBROWSE to Examine a SASHELP View

CATALOGS

Contents: Information about all currently allocated SAS catalogs. See Exhibit 4 for details.

A Row/Observation Is: A catalog entry (e.g., PROGRAM, SCL, KEYS)

Comments: The table resembles the output from the CONTENTS option of the CATALOG procedure. CATALOG, however, cannot process more than one catalog at a time. See "Comparability" in the next section for a comparison of the COLUMNS table and CONTENTS output datasets.

EXHIBIT 4: DICTIONARY CATALOGS, SASHELP.VCATALG

<table>
<thead>
<tr>
<th>LIBNAME</th>
<th>CHAR(8)</th>
<th>Label='Library Name'</th>
</tr>
</thead>
<tbody>
<tr>
<td>MEMNAME</td>
<td>CHAR(8)</td>
<td>Label='Member Name'</td>
</tr>
<tr>
<td>MEMTYPE</td>
<td>CHAR(8)</td>
<td>Label='Member Type'</td>
</tr>
</tbody>
</table>

EXHIBIT 4: DICTIONARY CATALOGS, SASHELP.VCATALG (CONTINUED)

<table>
<thead>
<tr>
<th>OBJNAME</th>
<th>CHAR(8)</th>
<th>Label='Object Name'</th>
</tr>
</thead>
<tbody>
<tr>
<td>OBJTYPE</td>
<td>CHAR(8)</td>
<td>Label='Object Type'</td>
</tr>
<tr>
<td>OBJDESC</td>
<td>CHAR(40)</td>
<td>Label='Object Description'</td>
</tr>
<tr>
<td>MODIFIED</td>
<td>CHAR(8)</td>
<td>Label='Date Modified'</td>
</tr>
<tr>
<td>ALIAS</td>
<td>CHAR(8)</td>
<td>Label='Object Alias'</td>
</tr>
</tbody>
</table>

EXHIBIT 5: DICTIONARY COLUMNS, SASHELP.VCOLUMN

LIBNAME  | CHAR(8) | Label='Library Name' |
MEMNAME  | CHAR(8) | Label='Member Name'  |
MEMTYPE  | CHAR(8) | Label='Member Type'  |
NAME     | CHAR(8) | Label='Column Name'  |
TYPE     | CHAR(4) | Label='Column Type'  |
LENGTH   | NUM    | Label='Column Length'|
NPOS     | NUM    | Label='Column Position'|
VARNUM   | NUM    | Label='Column Number in Table'|
LABEL    | CHAR(32)| Label='Column Label' |
FORMAT   | CHAR(16)| Label='Column Format' |
INFORMAT | CHAR(16)| Label='Column Informat' |
IOXUSAGE | CHAR(16)| Label='Column Index Type' |

EXHIBIT 6: DICTIONARY EXTFILES, SASHELP.VEXTFL

FILEREF | CHAR(8) | Label='Fileref' |
XPATH    | CHAR(8) | Label='Path Name' |
XENGINE  | CHAR(8) | Label='Engine Name' |

INDEXES

Contents: Information about indexes used by the currently allocated SAS datasets. See Exhibit 7 for details.

A Row/Observation Is: Each occurrence of a variable used in an index.

Comments: Some of the information in this table is also found in the CONTENTS output dataset. See Exhibit 15 for a comparison.

EXHIBIT 7: DICTIONARY INDEXES, SASHELP.VINDEX

LIBNAME | CHAR(8) | Label='Library Name' |
MEMNAME | CHAR(8) | Label='Member Name'  |
MEMTYPE | CHAR(8) | Label='Member Type'  |
NAME    | CHAR(8) | Label='Column Name'  |
INDEXUS | CHAR(8) | Label='Index Name'   |
INDNAME | CHAR(8) | Label='Index Name'   |
Contents: Information about all macro variables currently defined in the program. The table contains both automatic (SYSOATE, SYSSCP, etc.) and user-defined macro variables. The first 200 characters of the value are stored, along with the variable's scope (local, global, automatic). See Exhibit 8 for details.

A Row/Observation is: A member of a currently allocated SAS file.

Comments: The PATH variable identifies either a single operating system directory name or multiple names, if the LIBNAME was part of a series of directories in a search path.

MEMBERS

Contents: Information about all SAS files currently allocated in the batch job or interactive session. These files include SAS data sets, views, access descriptors, catalogs, and stored programs. See Exhibit 9 for details.

A Row/Observation is: A member of a currently allocated SAS file.

Comments: The PATH variable identifies either a single operating system directory name or multiple names, if the LIBNAME was part of a series of directories in a search path.

There are six SASHELP views which are row and column subsets of MEMBERS. The subsampling is described in Exhibit 10. To log your memory, remember that the views begin with "VS" (presumably, view subset).

Options:

Contents: Information about current settings for all SAS system options. See Exhibit 11 for details.

A Row/Observation is: A SAS system option.

Comments: The OPTIONS table and VOPTION view are good ways to review the meaning and current values of SAS system options. Unlike the Options Window in Display Manager, they also provide a description (variable OPTDESC) of each option.

Also unlike the window and OPTIONS procedure, the table and view permit options to be manipulated programmatically: you can retrieve the value of an option, store it in a macro variable, reset it, execute other statements, and then reset the option to its original value. This technique is especially useful for utility macros, since it enables them to alter system settings during the course of their execution and then reset them. Refer to Example 4 for an illustration of this technique.

TABLES

Contents: Information about currently allocated SAS data files. See Exhibit 12 for details.

A Row/Observation is: A SAS data file.

Comments: The table contains information about SAS data sets, views, and SCL tables. See "Comparability" in the next section for a comparison of the TABLES table and CONTENTS output datasets.

Titles:

Contents: Currently defined titles and footnotes. See Exhibit 13 for details.

A Row/Observation is: A title or footnote line.
Comments: This table is available only in Version 6.11 of the SAS System. The table has a maximum of 20 observations. The TYPE variable distinguishes titles (T) from footnotes (F).

**EXHIBIT 13: DICTIONARY.TITLES, SASHELP.VTITLE**

| NUMBER | num | label='Title Number' |
| TEXT | char(200) | label='Title Text' |
| TYPE | char(1) | label='Title Location' |

**VIEWS**

Contents: information about currently allocated SAS views. See Exhibit 14 for details.

**A Row/Rowset Id: A SAS view.**

Comments: The table contains information about PROC SQL views, dataset views, and SAS/ACCESS view descriptors.

**EXHIBIT 14: DICTIONARY.VIEWS, SASHELP.VVIEW**

| LIBNAME | char(8) | label='Library Name' |
| MEMNAME | char(8) | label='Member Name' |
| MEMTYPE | char(20) | label='Member Type' |
| ENGINE | char(8) | label='Engine Name' |

**COMMENTS AND USAGE NOTES**

Before presenting examples of how the tables may be used we present some general comments concerning enhancements, performance, and comparability.

**ENHANCEMENTS**

Once a tool becomes well-worn and familiar the user often wants to tinker with its design and scope. Such is the case with the dictionary tables. A table with system-specific items would be useful. It could contain the current user id, system settings (default directory, path, environmental variables, etc.), and other session-related information. Some of this is already available in AF functions, some in automatic macro variables, and some via system-specific functions. It would be helpful to have it all in one location.

More information in TABLES would also be helpful. For example, the size of the dataset could be stored. The number of storage units (tracks, blocks, bytes) for the entire dataset could be supplied — this is a step beyond simply multiplying NOBS by OBSLEN. If compression was used, a variable could contain the ratio of compressed to uncompressed. Items returned from the SCL functions ATTRN and ATTRC are also candidates for inclusion in TABLES.

**PERFORMANCE**

As the Introduction stated, the tables are not a "free lunch." Rather, they are a "low cost lunch." Still a bargain, but you need to be aware of some of the performance penalties surrounding their use. In order to get a feel for how performance of accessing tables compared to views we ran a series of benchmarks in several operating systems. The task was simple: retrieve all COLUMNS and TABLES information for several datasets, then do the same for their views (VCOLUMN and VTABLE). The access strategies are described below:

- **SQL Only.** Use only SQL to retrieve the data for the datasets of interest.
- **SQL, PROC.** Use SQL to create a table, then the PRINT procedure to display the information.
- **DATA, PROC.** Use a DATA step to create a dataset based on SASHELP views, then use the PRINT procedure to display the data.
- **View Only.** Use SASHELP views directly in a PRINT procedure.

The four test programs were run five times in each system. The last four runs' elapsed times were averaged. The results vary significantly across operating systems, and are presented in the graph in Exhibit 15. In all systems using the tables, either solely in PROC SQL or in SQL and a subsequent DATA step, yields consistently faster elapsed times. Once we use the SASHELP views in a DATA step performance suffers dramatically, although the difference between the view and table timings is less pronounced in the Unix system. The last test, using views directly in a PRINT procedure, produced the most varied timings. The Alpha and Windows timings were very slow compared to the table-oriented tests. OS/2 was only slightly slower and Unix was actually faster than the SQL timings.

It appears that an SQL-oriented approach to using the tables will, on average, produce time somewhat or significantly faster than methods using the SASHELP views. In some cases the selection of method is critical to the success of the application. Submitting a DATA step and procedure from a Windows AF application would require over 8 seconds to complete. Equivalent results could be obtained by submitting SQL and a procedure taking just over 2 seconds. The less time users of the system spend watching the Windows hourglass icon, the better.

**COMPARISONS WITH PROCEDURE OUTPUT DATASETS**

As we will see in the next section, one of the most appealing applications of the tables and views is generation of reports about SAS entities such as datasets, views, and catalogs. The "pre-table" way to do this outside the SASIAF environment was to generate output datasets from the CATALOG or CONTENTS procedures. The datasets would then be used in the PRINT or REPORT procedures, possibly supplemented by user-written formats to improve readability.

Dictionary tables often offer a more complete range of information about the entity in question. Since they can be used directly by the reporting procedure or DATA step they require less coding effort. You have to consider, however, what you want to report and if it's available in the tables. The following Exhibits compare items in catalog and dataset-related output datasets and tables. Exhibit 16 compares the information in a CATALOG procedure output dataset with that contained in the CATALOGS table.
The comparison is not nearly as straightforward when we compare
the CONTENTS output dataset to the dictionary tables. The
complexity arises in part because the CONTENTS output dataset
is not normalized. The unit of observation is the variable, but
each observation contains repeating information about the dataset as a
whole. This is convenient, but can render the task of gathering
similar table information difficult. Exhibit 17 looks at the output
dataset and the related dictionary tables. Please note that the com­
parisons are made for Version 6.10. CONTENTS variables may
vary on other platforms and versions of the SAS System.

Example 1: Displaying Values from SASHELP Views

The descriptions of the tables and views in the preceding section
have, by necessity, been brief. It is helpful to have a listing of both
the variables in each view and some of their observations. The
macro in Exhibit 18 creates this reference material. In the first
DATA step it counts the number of views associated with the
NAME of SASHELP. The DATA step also writes macro vari­
ables VIEW1-VIEWn, which contain the actual names of the views
(VTABLE, VCOLUMN, etc.). The %DO loop is executed for each
VIEW macro variable. It runs CONTENTS for the view, then prints
a user-controlled number of observations from the view. The VVIEW
view is the first DATA step accesses the views in alphabetical order, so
the listing is also produced alphabetically: VCOLUMN is processed first,
followed by VEXTFL and so on through VVIEW.

Example 2: Using Dataset Name and Label in a Print Utility

The program in Exhibit 19 puts the VTABLE view to work. It prints a
user-specified number of observations from a dataset. The title
associated with the PRINT procedure is the dataset name and label.
This is more informative than the default "The SAS System" title and
more reliable than hand-coding the information. Notice that we could
with relatively little effort put other useful, dataset-level information in
the title: number of observations, number of variables, creation date,
and other characteristics are available from the VTABLE view.
**ADVANCED TUTORIALS**

**Exhibit 19: Using VTABLE to Create Informative Titles**

```sas
%macro print-title(libname, memname);
  proc sql noprint;
    select label
      into :label
    from anas help.vtable
    where libname = "libname." & memname = "memname." & memtype in ("DATA", "VIEW")
  ;
quit;
proc print data=&libname •. &memname.;
title "Dataset &libname •. &memname. - &label.";
run;
%end;
```

**Example 3: Library-Level Dataset Reporting**

Project-related data is usually organized into distinct operating system files or directories. Clinical trials data for a particular compound, for example, may be in a directory (in VMS) such as "client:\compound_name.data.sasj". It would be helpful to have a compact listing of dataset characteristics (variable names, type, length, format, etc.) for all datasets in the library.

The REPORT code which follows uses the VCOLUMN view to create such a listing. Notice that the report is sorted by dataset name. Variations on this are easily created; you could sort by dataset name-variable name, or variable name only. Some pre-processing with a DATA step or SQL could create a subset of the data identifying variable names occurring more than once, variables using user-written formats, and the like. The code in Exhibit 23, at the end of this paper, is only a starting point for what could be a library of reports. The exhibit presents part of the output generated by this macro.

**Example 4: Using OPTIONS to Save System Settings**

An occasional, and unwanted, byproduct of using macros or included programs is that they reset system options. Aesthetic features such as centering or page numbering may be adjusted. Other, more critical, features may be reset to meet the needs of the program: options such as REPLACE, NOSNDFERR, and NOFMTERR can have a significant impact on execution of the "calling" program. One obvious solution is to know exactly what goes on in the subroutine code. Many factors - time constraints, unpublicized revisions of working code, and the like - conspire to make complete knowledge an elusive goal.

The macro in Exhibit 20 uses the OPTIONS table to remember and restore the value of several system options. The macro requires OPTIONS CENTER and DATE to be turned off, so it obtains those values from the Dictionary Table, sets them (the OPTIONS statement), runs a procedure, then resets CENTER and DATE to their original values prior to terminating.

**Exhibit 20: Save/Restore Values of System Options**

```sas
%macro util1;
  proc sql noprint;
    select setting into :tablcent
      from dictionary.options
      where optname="center";
    select setting into :tabldate
      from dictionary.options
      where optname="date";
  quit;

  options nodate nocenter;
  ... proc go here ... * Reset to original values
  options &tablcent. &tabldate.;
%mend;
```

**Example 5: Moving Command Procedure Code into the SAS Program**

An applications programmer will typically not let SAS code exist in a vacuum, unaffected by the host operating system. Systems usually have a command language which performs tasks more efficiently and easily than would be possible in the SAS System. The VAX command language, DCL, for example has numerous functions which allow the programmer to test for the presence of files, carry on a dialog with the user, and other tasks.

The dictionary tables do not replace all of this functionality. However, the information they contain may encourage some programmers to move activities previously reserved for system command languages into SAS code. This makes programs more self-contained and thus more portable across platforms.

A before-and-after example is presented in Exhibit 21. Output generated from procedures needs to be routed to the user's SASUSER directory. Since the path is system dependent, code specific to the host system can be used to retrieve the name of the directory (top portion of the Exhibit). However, using dictionary tables, the path can be obtained without using code specific to the host (bottom portion of the Exhibit). This makes the application more portable.

**Exhibit 21: Moving Command Language Code into the SAS System**

```sas
"Before" - Using Operating-System Specific Code
x vms 'mydir = $sysget(mydir)';
%let path = $sysget(mydir);
proc printto print="&path.myfile.lis";

"After" - Using Dictionary Tables
proc sql noprint;
  select path into :path
    from dictionary.members
    where libname = 'SASUSER';
%let path = qtrim(&path.);
proc printto print="&path.myfile.lis";
```

**Example 6: Counting Observations in a Dataset**

Decision-making during macro execution is sometimes based on the number of observations in a dataset. There are many ways to do this. A simple example is shown here because the TABLES table provides a fairly clean method for answering two questions simultaneously: does the dataset exist, and if so, how many observations does it contain? The macro NINDSET is found in Exhibit 22. The user specifies the name of the dataset using the standard two-level name notation.

NINDSET creates macro variable COUNT. It is -1 if the dataset could not be located, 0 or higher otherwise (indicating the number of observations in the dataset).

**Exhibit 22: Counting Observations with TABLES Table**

```sas
%macro nindset(data);
  %let data = %upcase(&data.);
  %global count;
  %let count=-1;
  %if %str(&data.)*
    %then %let count=-1;
    %let libname = %scan(&data., 1, ".");
    %letdsnname = %scan(&data., 2, ".");
    %let ldbname = %str(&libname.");
    %let dsname = %str(&dsnname.);
    %let libname = %upcase(&libname.);
    %let dsname = %upcase(&dsnname.);
    %letmtime = %put;
    %end;
```

**218**
**Exhibit 23: Observation Counting with TABLES Table (continued)**

```sql
proc sql noprint;
select nobs
from sashelp.nodes
where memname="s1dbname." &
libname = "s1dbname." &
execname = 'DATA';
quit;
```

**Exhibit 23: Use PROC REPORT and VCOLUMN to create library-level report**

```sql
macro lib_rept(libname=WORK);
%let libname = %uppercase(libname);
proc report panels=3 pspace=2 spacing=1 headline
column memname name type
&libname;
%let libname : %upcase(&libname.);
data=sashelp.vcolumn
colnum=1
nowindow
nlayer;
run;
%mend lib_rept;
```

**Partial Output:**

<table>
<thead>
<tr>
<th>Member</th>
<th>Name</th>
<th>Name</th>
<th>Type Len. Format</th>
<th>Member</th>
<th>Name</th>
<th>Name</th>
<th>Type Len. Format</th>
</tr>
</thead>
<tbody>
<tr>
<td>ADVERSE</td>
<td>PTID</td>
<td>char</td>
<td>4 6.25</td>
<td>IVV</td>
<td>char</td>
<td>4 5.4</td>
<td></td>
</tr>
<tr>
<td></td>
<td>OD</td>
<td>char</td>
<td>4 5.4</td>
<td>PAT</td>
<td>char</td>
<td>3 1.3</td>
<td></td>
</tr>
<tr>
<td></td>
<td>TREAT</td>
<td>char</td>
<td>5 5.5</td>
<td>REC_NIM</td>
<td>num</td>
<td>3 1.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>ADXL</td>
<td>char</td>
<td>60 56</td>
<td>VISRUM</td>
<td>num</td>
<td>7 8</td>
<td></td>
</tr>
<tr>
<td></td>
<td>ASEXST</td>
<td>char</td>
<td>60 46</td>
<td>VIRE</td>
<td>num</td>
<td>8 DATE7</td>
<td></td>
</tr>
<tr>
<td></td>
<td>BESCT</td>
<td>char</td>
<td>40 46</td>
<td>TIME1</td>
<td>num</td>
<td>7 TIMES5</td>
<td></td>
</tr>
<tr>
<td></td>
<td>ONSET</td>
<td>num</td>
<td>8 DATE7</td>
<td>STYVOT</td>
<td>num</td>
<td>8 DATE7</td>
<td></td>
</tr>
<tr>
<td></td>
<td>7 TIMES5</td>
<td>num</td>
<td>7 TIMES5</td>
<td>TIMES1</td>
<td>num</td>
<td>7 TIMES5</td>
<td></td>
</tr>
<tr>
<td></td>
<td>ESTDLOGO</td>
<td>char</td>
<td>23 63</td>
<td>60</td>
<td>num</td>
<td>8 10.2</td>
<td></td>
</tr>
<tr>
<td></td>
<td>SEVERITY</td>
<td>char</td>
<td>10 51.3</td>
<td>60</td>
<td>num</td>
<td>8 10.2</td>
<td></td>
</tr>
<tr>
<td></td>
<td>VISCHOL</td>
<td>char</td>
<td>3 8.1</td>
<td>60</td>
<td>num</td>
<td>8 10.2</td>
<td></td>
</tr>
<tr>
<td></td>
<td>TREAT</td>
<td>char</td>
<td>3 8.3</td>
<td>60</td>
<td>num</td>
<td>8 10.2</td>
<td></td>
</tr>
<tr>
<td></td>
<td>DONOGEO</td>
<td>char</td>
<td>3 1.1</td>
<td>60</td>
<td>num</td>
<td>8 10.2</td>
<td></td>
</tr>
<tr>
<td></td>
<td>COMF416C</td>
<td>char</td>
<td>3 1.1</td>
<td>60</td>
<td>num</td>
<td>8 10.2</td>
<td></td>
</tr>
<tr>
<td></td>
<td>ADEXT1</td>
<td>char</td>
<td>200 $2000</td>
<td>60</td>
<td>num</td>
<td>8 10.2</td>
<td></td>
</tr>
<tr>
<td></td>
<td>ADEXT2</td>
<td>char</td>
<td>200 $2000</td>
<td>60</td>
<td>num</td>
<td>8 10.2</td>
<td></td>
</tr>
<tr>
<td></td>
<td>FROT</td>
<td>char</td>
<td>4 6.4</td>
<td>60</td>
<td>num</td>
<td>8 10.2</td>
<td></td>
</tr>
<tr>
<td></td>
<td>INVTD</td>
<td>char</td>
<td>4 5.4</td>
<td>60</td>
<td>num</td>
<td>8 10.2</td>
<td></td>
</tr>
<tr>
<td></td>
<td>FAT</td>
<td>char</td>
<td>3 5.3</td>
<td>60</td>
<td>num</td>
<td>8 10.2</td>
<td></td>
</tr>
<tr>
<td></td>
<td>REC_NIM</td>
<td>num</td>
<td>3 5.3</td>
<td>60</td>
<td>num</td>
<td>8 10.2</td>
<td></td>
</tr>
<tr>
<td></td>
<td>VISRUM</td>
<td>char</td>
<td>7 67</td>
<td>60</td>
<td>num</td>
<td>8 10.2</td>
<td></td>
</tr>
<tr>
<td></td>
<td>TIME</td>
<td>num</td>
<td>8 DATE7</td>
<td>60</td>
<td>num</td>
<td>8 10.2</td>
<td></td>
</tr>
<tr>
<td>BASEDX</td>
<td>PTID</td>
<td>char</td>
<td>7 57</td>
<td>60</td>
<td>num</td>
<td>8 10.2</td>
<td></td>
</tr>
<tr>
<td></td>
<td>OD</td>
<td>char</td>
<td>4 6.4</td>
<td>60</td>
<td>num</td>
<td>8 10.2</td>
<td></td>
</tr>
<tr>
<td></td>
<td>TREAT</td>
<td>char</td>
<td>5 5.5</td>
<td>60</td>
<td>num</td>
<td>8 10.2</td>
<td></td>
</tr>
<tr>
<td></td>
<td>TYPB</td>
<td>char</td>
<td>1 81</td>
<td>60</td>
<td>num</td>
<td>8 10.2</td>
<td></td>
</tr>
<tr>
<td></td>
<td>MUNIND</td>
<td>char</td>
<td>40 $200</td>
<td>60</td>
<td>num</td>
<td>8 10.2</td>
<td></td>
</tr>
<tr>
<td></td>
<td>SLOT</td>
<td>char</td>
<td>1 81</td>
<td>60</td>
<td>num</td>
<td>8 10.2</td>
<td></td>
</tr>
</tbody>
</table>