The Fast Food Approach to Data Warehouse Reporting: Using SAS/AF® and FRAME to Build a Non-Technical User Interface

Grace La Torra, New Mexico State University, Las Cruces, N.M.
Esther D. Steiner, New Mexico State University, Las Cruces, N.M.

Abstract

Many data warehouse implementations are hampered by the inability of inexperienced users to construct accurate queries. Users must either rely upon canned reports, or upon the whims and availability of programmers. The development of a graphical user interface (GUI) as a user tool can help alleviate these problems by giving the user power to develop his or her own queries without extensive knowledge of the structure of the data.

This paper describes the development and implementation of a GUI front-end user interface called TABLES with SAS/AF® and SAS Screen Control Language (SCL). This interface is Windows compatible. TABLES allows the user to access a PC-based data warehouse of employee related data maintained in SAS libraries. By choosing from the available options, the user can select and customize reports, including titles and footnotes. TABLES provides power and flexibility for the inexperienced user, while freeing programmers to spend more time programming.

Introduction

We are programmers trapped in a non-programming world. A large part of our work involves providing ad hoc reports of many different, but related, flavors. Over the course of the last year we have designed and implemented a GUI that translates user selections from button and list selections into SAS code, using SQL and PROC TABULATE. This has greatly decreased the amount of time that programmers spend rewriting reports.

Why Did We Do This?

Our office was inundated with requests, each varying slightly from the others, for information on employee counts and characteristics.

We have attempted to address this problem over the years through a number of methods. We began by printing reports for every information request we anticipated we could possibly have. Not only had these reports grown to an immense size, but more often than not the information needed *this time* was not included, and yet another report would have to be added, requiring further programming. Next we tried creating a program that included every possible field and table, which could then be customized by removing unneeded data. However, this still required programmer intervention to customize the program for each request.

Faced with these problems, we wanted to maintain flexibility while still reducing the amount of time programmers spent maintaining and developing reports. To this end, we began developing a point and click interface for the non-technical users.

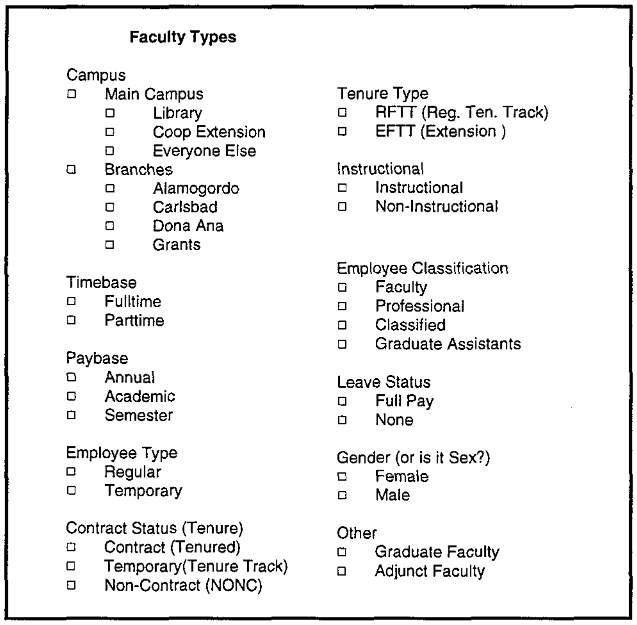
What Is It *REALLY*?

We were already using SAS® for Windows 6.10 on several PC Pentium® processors. These computers are networked together using Windows for Workgroups, with the data residing on a Windows® NT Server. We were not interested in introducing any new hardware or software to meet our needs.

The data are stored in a data warehouse, code-named ODYSSEY. The data come from several different mainframe sources, including the university's personnel/payroll system, the financial records system, and the student records system. Data are downloaded from the mainframe each semester in ASCII files, validated for accuracy and consistency, and then loaded into permanent SAS data sets. There are 15 data sets, encompassing records for approximately 10,000 employees per semester. Additionally, there are 30 supporting data sets containing data which varies little from semester to semester.

We chose to develop the TABLES program in SAS/AF because of the availability of widgets with which to build a user interface. Widgets are predefined objects with specific characteristics and operations. Examples of widgets commonly seen in Windows programs include radio buttons, check boxes, list boxes, push buttons, and text boxes. A frame is a screen containing widgets, and may also have supporting code, called Screen Control Language (SCL), behind it.

We began developing the TABLES program by identifying the different ways in which employees could be defined. Originally, we had used a check sheet (Figure 1) for processing requests. This request form was used as the basis for the program.



Faculty Types

Campus
- ☐ Main Campus
  - ☐ Library
  - ☐ Coop Extension
  - ☐ Everyone Else
- ☐ Branches
  - ☐ Alamogordo
  - ☐ Carlsbad
  - ☐ Dona Ana
  - ☐ Grants

Timebase
- ☐ Fulltime
- ☐ Parttime

Paybase
- ☐ Annual
- ☐ Academic
- ☐ Semester

Employee Type
- ☐ Regular
- ☐ Temporary

Contract Status (Tenure)
- ☐ Contract (Tenured)
- ☐ Temporary(Tenure Track)
- ☐ Non-Contract (NONC)

Tenure Type
- ☐ RFTT (Reg. Ten. Track)
- ☐ EFTT (Extension)

Instructional
- ☐ Instructional
- ☐ Non-Instructional

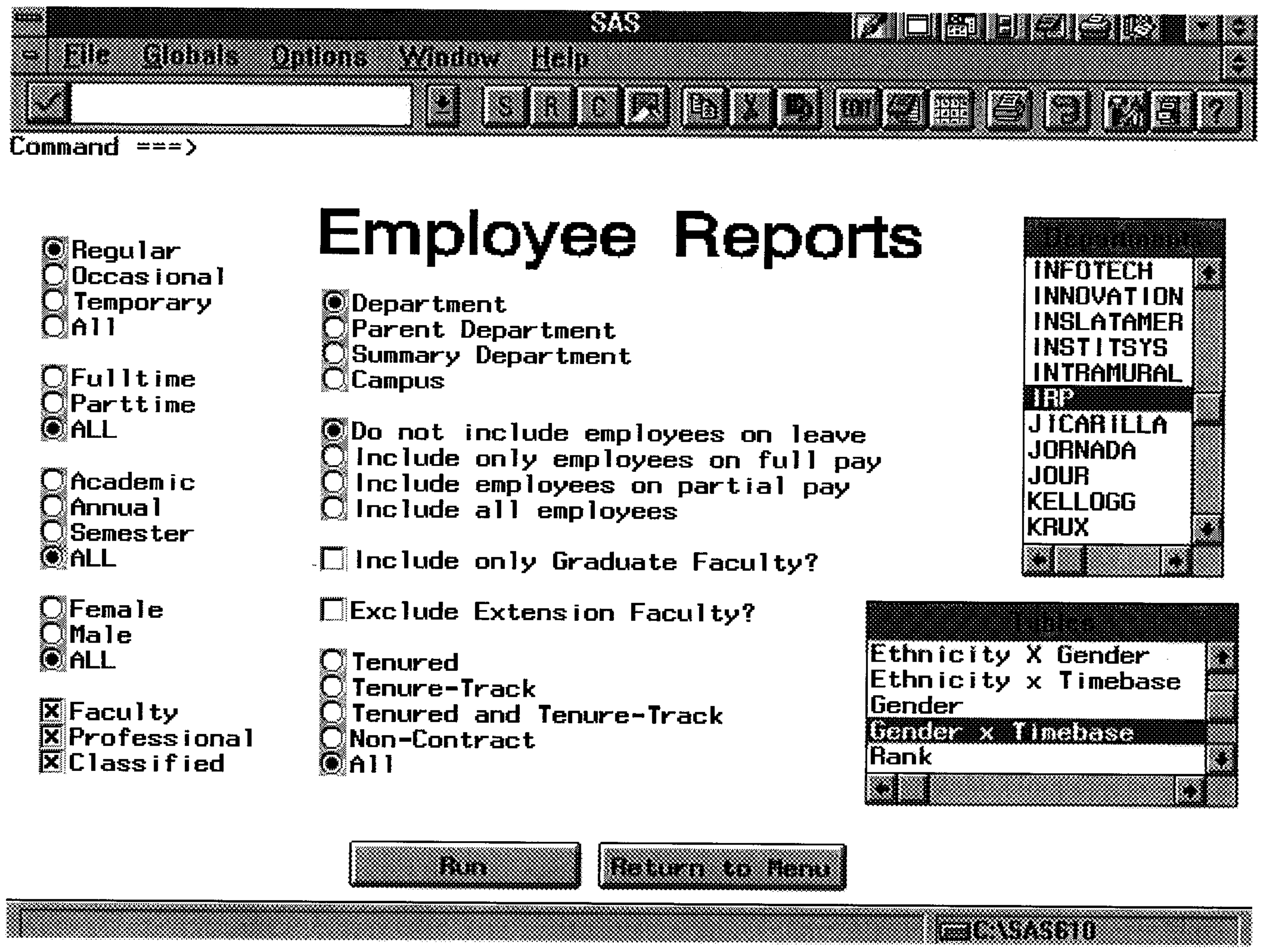
Employee Classification
- ☐ Faculty
- ☐ Professional
- ☐ Classified
- ☐ Graduate Assistants

Leave Status
- ☐ Full Pay
- ☐ None

Gender (or is it Sex?)
- ☐ Female
- ☐ Male

Other
- ☐ Graduate Faculty
- ☐ Adjunct Faculty

Figure 1.



Employee Reports

[✓ Regular]
Occasional
Temporary
All

Fulltime
Parttime
[✓ ALL]

Academic
Annual
Semester
[✓ ALL]

Female
Male
[✓ ALL]

[✓ Faculty]
[✓ Professional]
[✓ Classified]

[✓ Department]
Parent Department
Summary Department
Campus

[✓ Do not include employees on leave]
Include only employees on full pay
Include employees on partial pay
Include all employees

Include only Graduate Faculty?

Exclude Extension Faculty?

Tenured
Tenure-Track
Tenured and Tenure-Track
Non-Contract
[✓ All]

INFOTECH
INNOVATION
INSLATAMER
INSTITSYS
INTRAMURAL
[✓ IRP]
JICARILLA
JORNADA
JOUR
KELLOGG
KRUX

Ethnicity X Gender
Ethnicity x Timebase
Gender
[✓ Gender x Timebase]
Rank

Figure 2.

We began by identifying the different categories by which employees could be described. We then decided which categories were best displayed with check boxes or radio buttons. Check boxes were used for those selections in which a choice was not exclusive, and radio buttons were used in those cases in which only one selection was appropriate. In some cases, there were too many choices for either check boxes or radio buttons to be appropriate, so list boxes were used instead. At this point our TABLES frame looked like Figure 2.

Behind the Scenes

Behind the TABLES frame we wrote SCL code which checked the values of each widget, and then constructed the appropriate SAS code to generate the desired report. SCL has built in functions that can be used to obtain the widget values. These include _GET_TEXT_ and _GET_VALUE_. The widget values were then placed in macro variables that can be used at any point within SAS code. In our case, we used them to construct the WHERE clause in a SQL query. This SQL query builds the final data set for use in reporting. The reports themselves were produced using PROC TABULATE, where once again the CLASS, TABLE, and FORMAT clauses were constructed using the macro variables. The TABLES SCL code is shown in Appendix I.

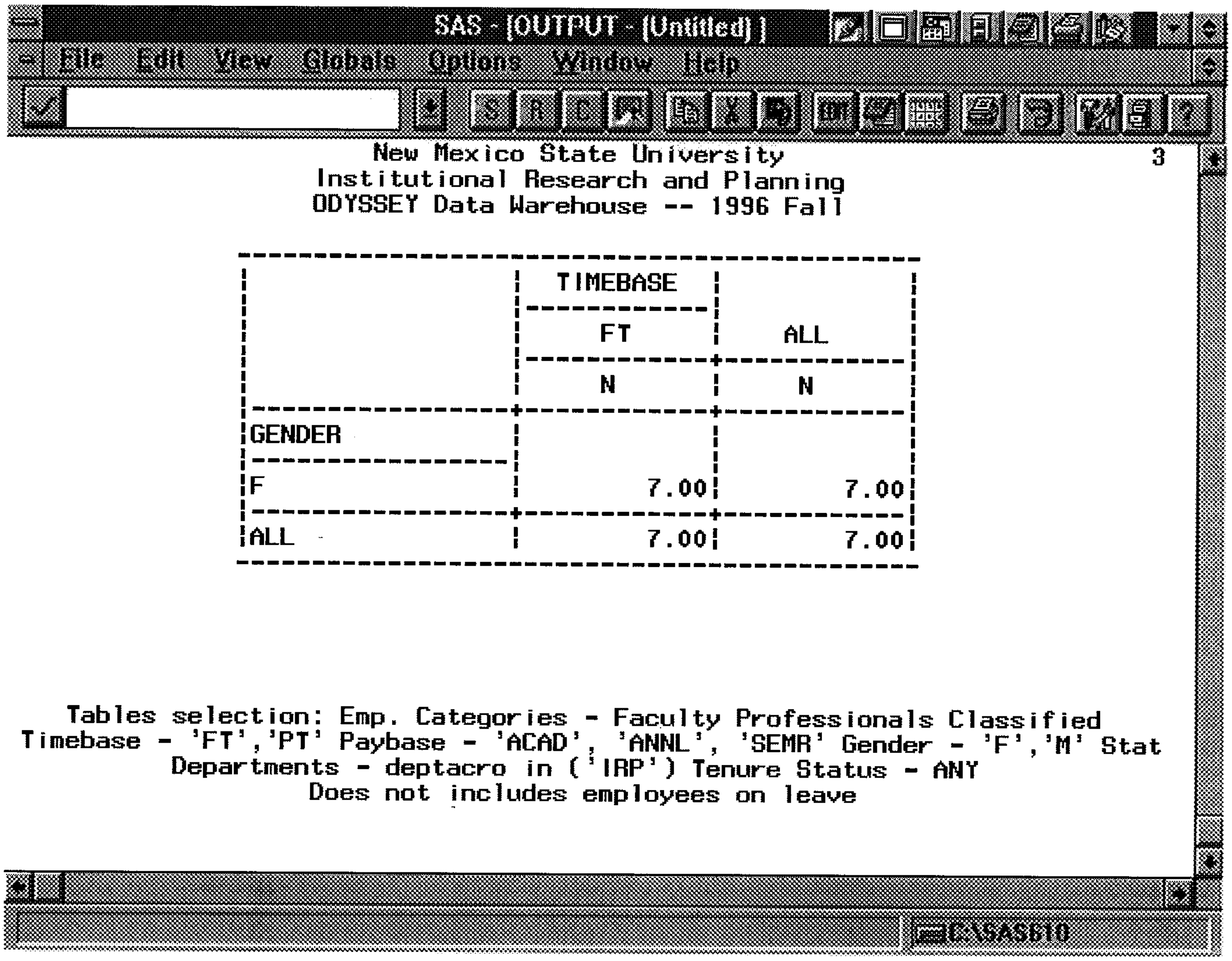
The list box widgets for Department and Tables are populated with the records from SAS data sets. New departments can be added to the department data set and these will automatically display as the frame is constructed. Similarly, the names of all available reports and the SAS code needed to create the reports are maintained in a data set called TABLIST. These report names are also displayed in the list box as the frame is constructed.

When a user selects a table, the appropriate SAS code is retrieved from the TABLIST data set. Through the use of macro variables, this code is then included in a PROC TABULATE statement and submitted to the SAS interpreter. In the interest of saving time, the Tables list box allows the user to make multiple selections before submitting the code for execution.

As a part of the TABLES process, the SCL also constructs footnotes detailing the selections made by the user. This allows the user to reproduce a given report and also provides a means of documenting the report contents. An example of a report produced by the TABLES program is shown in Figure 3.

Benefits

One significant benefit of this approach is the complete lack of any hard-coding. New reports can be generated without any further investment in programming. This empowers the user to generate his or her own reports and frees the programmer for other tasks.


SAS - [OUTPUT - (Untitled)]
File Edit View Globals Options Window Help
C:\SAS610

```
                    New Mexico State University                    3
                  Institutional Research and Planning
                  ODYSSEY Data Warehouse -- 1996 Fall

          -----------------------------------------------
          |                 |  TIMEBASE  |            |
          |                 |------------|            |
          |                 |     FT     |    ALL     |
          |                 |------------+------------|
          |                 |     N      |     N      |
          |-----------------+------------+------------|
          |GENDER           |            |            |
          |-----------------|            |            |
          |F                |        7.00|        7.00|
          |-----------------+------------+------------|
          |ALL              |        7.00|        7.00|
          -----------------------------------------------



     Tables selection: Emp. Categories - Faculty Professionals Classified
  Timebase - 'FT','PT' Paybase - 'ACAD', 'ANNL', 'SEMR' Gender - 'F','M' Stat
           Departments - deptacro in ('IRP') Tenure Status - ANY
                     Does not includes employees on leave
```

Figure 3.

TABLES is also easy to maintain. New choices can be easily added to existing widgets in the frame. No editing of the SCL code is required to accomplish this.

Conclusion

Use of TABLES within our office has decreased the amount of time that programmers must spend responding to ad hoc reporting requests. These reporting requirements can now be met using non-technical staff. Reports are self-documenting, with the selection criteria automatically included in the footnotes.

References

SAS Institute Inc. (1989), *SAS/AF® Software: Usage and Reference, Version 6, First Edition,* Cary, NC: SAS Institute Inc.

SAS Institute Inc. (1993), *SAS/AF® Software: FRAME Entry Usage and Reference, Version 6, First Edition,* Cary, NC: SAS Institute Inc.

SAS Institute Inc. (1993), *SAS® Companion for the Microsoft Windows Environment, Version 6, First Edition,* Cary, NC: SAS Institute Inc.

SAS Institute Inc. (1990), *SAS® Guide to Macro Processing, Version 6, Second Edition,* Cary, NC: SAS Institute Inc.

SAS Institute Inc. (1989), *SAS® Guide to the SQL Procedure usage and Reference, Version 6, First Edition,* Cary, NC: SAS Institute Inc.

SAS Institute Inc. (1990), *SAS® Screen Control Language, Reference, Version 6, First Edition,* Cary, NC: SAS Institute Inc.

SAS Institute Inc. (1991), *SAS® Screen Control Language, Usage, Version 6, First Edition,* Cary, NC: SAS Institute Inc.

Grace La Torra, NMSU, Box 30001 Dept. 3004, Las Cruces, NM, 88003-8001, (505) 646-6286, grace@nmsu.edu

Esther D. Steiner, NMSU, Box 30001 Dept CS, Las Cruces, NM, 88003-8001, (505) 646-2096 esteiner@cs.nmsu.edu

Appendix I

```
/****************************************************************
TABLES.SCL -- This program produces the faculty reports via a
screen  selection.

Copyright 1996 by the Regents of New Mexico State University.


Grace La Torra, Institutional Research and Planning.
January, 1996.
****************************************************************/



INIT:

/*** Initialize SCL macro variables. ***/

  length time2 $ 10;
  length pay2 $ 25;
  length tenstat2 $ 32;
  length gender2 $ 10;
  length status2 $ 15;
  length extflag2 $ 25;
  length gradflg2 $ 35;
  length leavst2 3;
  length deptlist  200;
  length tablist2 200;
  length nseld 5;
  length nselt 5;
  length deptsel $ 15;
  length fasel $ 10;
  length prsel $ 10;
  length clsel $ 10;
  length run $ 10;
  length foottext $ 160;
  length foot2 $ 160;
  length foot3 $ 160;
  length foot4 $ 160;

  submit continue;

/*** Create department and Tables list for display on screen. ***/


    proc sql noprint;

      create table depart as
        select deptacro
          from suppsems.depart
          order by deptacro;

      create table tablist as
        select tabname
          from support.tablist
          order by tabname;

/*** Build an employee table containing all the information  ***/
/*** that MIGHT be needed.                                    ***/

      create table emps as
      select *, '  ' as degcode, 0 as hireyrs, hiredate,
          ansalary as acadsal
        from all.personal,
          empsem.perdata,
          empsem.job,
          empsem.jobtitle,
          suppsems.depart
        where personal.ssn = job.ssn
          and perdata.ssn = job.ssn
          and job.jobcode = jobtitle.jobcode
          and job.deptacro = depart.deptacro;

      update emps
        set acadsal = acadsal * .75
        where paybase = 'ANNL';

      update emps
        set degcode = (select min(degcode)
                    from empsem.degree,
                      support.degname
                    where emps.ssn = degree.ssn
                      and degree.degree = degname.degree
                      and highdeg = 'Y');

      create table emps3 as
        select emps.*, degree, degdate
          from emps left join
            empsem.degree
          on emps.ssn = degree.ssn
          where highdeg = 'Y';

      drop table emps;

      create table emps2 as
      select emps3.*, tenstat, tendate, ttdate,
          tenure.deptacro as tendept, extflag
        from emps3 left join empsem.tenure
        on emps3.ssn = tenure.ssn;

      drop table emps3;

      create table emps as
      select *
        from emps2 left join empsem.rank
        on emps2.ssn = rank.ssn;

      drop table emps2;

    quit;

   data empssel;
     set empssel;
     hired2 = substr(hiredate, 3, 2);
     hireyrs = 95 - hired2;

   run;

  endsubmit;

return;



MAIN:

  call notify ('OK', '_get_text_', run);

/*** Run in this section so multiple reports can be created. ***/

if run = 'RUN' then do;

/*** Ask the Frame widgets for their present values. ***/

  call notify ('timebase', '_get_text_', time2);
  call notify ('paybase', '_get_text_', pay2);
  call notify ('gender', '_get_text_', gender2);
  call notify ('tenstat', '_get_text_', tenstat2);
  call notify ('status', '_get_text_', status2);
  call notify ('extflag', '_get_text_', extflag2);
  call notify ('gradflag', '_get_text_', gradflg2);
  call notify ('leavstat', '_get_value_', leavst2);
  call notify ('deptacro', '_get_value_', deptlist);
  call notify ('deptacro', '_get_nselect_', nseld);
  call notify ('tablist', '_get_value_', tablist2);
```

```
   call notify ('tablist', '_get_nselect_', nselt);
   call notify ('deptchck', '_get_text_', deptsel);
   call notify ('fa', '_get_text_', fasel);
   call notify ('pr', '_get_text_', prsel);
   call notify ('cl', '_get_text_', clsel);

/*** Now for the fun -- since the Department List box allows    ***/
/*** multiple entries, we have to step through, finding each    ***/
/*** value, and then build a coherent line of code from them.   ***/

   dept4 = "' ' in (' ')";
   if nseld > 0 then do;
     dept2 =  getiteml(deptlist,1);
     dept3 = getitemc(dept2, 1);
     dept4 =   deptsel || " in ('" || dept3;

     do i=2 to nseld;
       dept3 = getitemc(dept2, i);
       dept4 = dept4 || "', '" || dept3;
     end;
     dept4 = dept4 || "')";
   end;

/*** Set a default table in case none was selected, then do the  ***/
/*** same thing for the tables that we just did for the          ***/
/*** departments. However in this case, we must not only find    ***/
/*** the selected table values, we must go get the code out of   ***/
/*** the TABLIST dataset in the Support library.  Rather than    ***/
/*** using a datastep or sql, we use SCL.  Yet another way to    ***/
/*** get data!                                                   ***/

   if nselt = 0 then
     tab4 = 'class gender rank; Table Gender all; table rank all;';
   else do;
     tabs = open('support.tablist','i');
     vnum=varnum(tabs, 'tabdesc');
     tab2 = getiteml(tablist2,1);
     tab4 = ' ';
     do i=1 to nselt;
       tab3 = getitemc(tab2, i);
       rc = where(tabs, "tabname = " || quote(tab3));
       rc = fetch(tabs);
       if (rc ^= 0) then _msg_=sysmsg();
       tabdesc = getvarc(tabs, vnum);
       tab4 = tab4 || tabdesc;
     end;
     rc = close(tabs);
   end;

/*** Here's a nifty one!  We found that just printing the output   ***/
/*** without  documentation of what went into it is VERY           ***/
/*** confusing.  To remedy this problem we now automatically       ***/
/*** generating footnotes that document the user selections.       ***/
/*** Basically, the same macro variables that will be used in      ***/
/*** the SQL select statement are placed into the footnotes.       ***/

   foottext = 'Tables selection: Emp. Categories -';
   if fasel ^= "' ','" then foottext = foottext || ' Faculty';
   if prsel ^= "' ','" then foottext = foottext || ' Professionals';
   if clsel ^= "' '" then foottext = foottext || ' Classified';
   foot2 = ' Timebase - ' || time2;
   foot2 = foot2 ||' Paybase - ' || pay2;
   foot2 = foot2 || ' Gender - ' || gender2;
   foot2 = foot2 || ' Status - ' || status2;
   if dept4 ^= "' ' in (' ')" then
     foot3 = ' Departments - ' || dept4;
   else foot3 = ' Departments - ALL ';
   if tenstat2 ^= "' ' in (' ')" then
     foot3 = foot3 || ' Tenure Status - ' || substr(tenstat2, 10, 22);
   else foot3 = foot3 || ' Tenure Status - ANY ';
   if extflag2 ^= "' ' in (' ')" then
     foot3 = foot3 || ' Extension Flag - ' || extflag2;
   if gradflg2 ^ = "' ' in (' ')" then
     foot3 = foot3 || ' Graduate Flag - ' || gradflg2;
```

```
     if leavst2 = -1 then
       foot4 = ' Includes all employees including those on leave';
     else if leavst2 = 0  then
       foot4 = ' Includes employees on leave with partial pay';
     else if leavst2 = 99  then
       foot4 = ' Includes employees on leave with full pay';
     else if leavst2 = 100  then
       foot4 = ' Does not includes employees on leave';

    submit continue status;

      proc sql;

/*** Now for the real fun -- select only those employees that   ***/
/*** meet the specified criteria by including macro variables   ***/
/*** that will resolve at execution time.                       ***/

        create table empssel as
          select * from emps
            where empcat in (&fasel &prsel &clsel)
              and timebase in (&time2)
              and paybase in (&pay2)
              and gender in (&gender2)
              and status in (&status2)
              and (leavpay = . or leavpay > &leavst2)
              and &dept4
              and &tenstat2
              and &extflag2
              and &gradflg2;

        update empssel
          set fte = fte / 100;

/*** Assign the newly created footnotes. ***/

footnote1 "&foottext";
footnote2 "&foot2";
footnote3 "&foot3";
footnote4 "&foot4";

      quit;

/*** And run the tables!  Note that a proc tabulate can contain  ***/
/*** multiple table statements, so the user can send several at  ***/
/*** once.                                                       ***/

      proc tabulate data = empssel;
        &tab4;

      run;

    endsubmit;

  end;

return;




TERM:

/*** They must be done!  FINALLY! ***/

   if _status_ = 'C' then do;

/*** Clean up a little before finishing.  Most importantly, put ***/
/*** the footnotes back to how they were before!               ***/

     submit continue;

       footnote1 ' ';
      proc sql;

        drop table emps;
        drop table empssel;
        drop table depart;
        drop table tablist;

      quit;

    endsubmit;
    return;

  end;

return;
```