

Cross Platform Data Warehouse Access, Documentation, and Shared Object Functionality Using the SAS® System.

Thomas M. Skinner - Intel Corporation, Folsom, CA

ABSTRACT

This paper will summarize the methods used to successfully design and implement a SAS based system for managing access to a corporate Data Warehouse distributed across an IBM® MVS/ESA® based DB2® and NCR® (Now AT&T®) UNIX SVR4® based Sybase System 10® Relational Database Management Systems (RDBMS's). Topics include: Using the Metadata repository to generate SAS/ACCESS® Descriptors, Views, and Source Entries; Incorporation of Naming Standards and the inherent advantage of these in a central library of data objects in promoting shared code objects; Migration and portability considerations; Technical Architecture considerations; and End User Interface and Training Issues.

INTRODUCTION

While the concept of the corporate data warehouse is not a new one, the paradigm shift from a centralized mainframe source of data to distributed server based databases presents new challenges for database administrators and developers alike. As is often the case, our mainframe resources are a mix of RDBMS tables, extracts from these in the form of flat files, VSAM files and SAS data sets. Access to any of the above usually required some familiarity with the source, as well as necessitating acquiring security clearances and permissions, and inevitably resolving difference in coding practices and file format layouts. The data warehouse concept was created out of a need to streamline this process and provide a consistent source of data in a reliable manner.

Among the key elements in the successful implementation of a data warehouse are: accessible, reliable and accurate metadata; a good data model which fits a wide range of business requirements; support and participation of both the key contributors of the data, i.e. the various MIS groups within each business unit, and from the end user programmers in terms of defining what constitutes the right data to be included in the data warehouse, and specifying how the data

was to be made available in an efficient and timely manner.

Given these elements, I was presented with a fertile ground for planting the seeds which would grow into the this system for end user programmer access while simultaneously enabling the promotion of a corporate asset in the form of a common software library based on SAS code.

While the initial application of the techniques described in this paper were made in the mainframe environment, the design of the data warehouse was such that cross platform compatibility and transportability of applications was accounted for. In particular, the mirroring of table structures both directly, and through the use of views, enabled transparent porting of both the system described herein as well as end user applications.

The initial phase of development on the mainframe platform has been code named FAST TRACK and consists of carefully modeled DB2 tables designed specifically for Decision Support Systems (DSS). While access was granted to these tables for general reporting needs, they are served a dual role in providing firstly a staging area for data to be transported to distributed servers. At the time of this writing, there were over 300 tables in the FAST TRACK databases. This platform was designed to enable end user programmers to begin development of applications which will go against RDBMS structured data.

The long term goal is to provide solutions for well defined business needs for data on geographically distributed server platforms based on vertically integrated Intel architecture based systems, including the desktop, LAN servers, and data servers ranging from single to multiple processor "super-servers". It is this architecture that will ultimately support the "Data Warehouse".

The move to adopting a data warehouse generally involves a radical shift in focus. In fact, the driving force for implementing a distributed data warehouse often involves more than a perceived benefit based on Client/Server technologies, in

that entire business process may be re-engineered. For example, routines which once generated hardcopy reports carried out onto the shop floor, are now modified to provide summarized data which feeds touch screen based Executive Information Systems with extensive drill down capabilities and significant post-processing analytical capabilities. It is with this in mind, that the concentration of the effort be aimed at new system development, while adapting currently essential Legacy systems to the new sources of data.

Data Warehouse and the Common Software Library

My initial involvement in the data warehouse project was as a technical resource helping an organization having significant experience and investment in SAS programming, develop routines designed to provide data to the data warehouse. This involved developing production worthy techniques for extracting data and preparing it for transport. I was soon invited to participate in both the Technical Architecture Group involved in the design of the technical capabilities of the data warehouse, and later, the End User Programmer Steering Committee, charged with ensuring the suitability of the data warehouse in meeting the widely ranging requirements of all of the business units in the company.

Concurrent to this activity, the Tools and Methods Integration group, of which I am a member, was charged with the setting of standards, methods, and management of a shared common software library for use in application development. Traditional application development had been primarily the responsibility of individual business units resulting in much overlap and poor coordination. Not only did this cause inefficiencies in computing resources, it also resulted in much duplication of programming effort as well. It is through the common software library, building off of a solid foundation provided by the distributed data warehouse that we intend to remedy this inefficiency.

A common base of standard data object names for systems developed in SAS language, be it base language or SCL, when combined with solid Object Oriented analysis and design techniques, will provide the essential elements required for a successful implementation of interchangeable and re-usable code library. In this way, application development becomes more a matter of integration rather than the labor intensive method of code from scratch. Each element so developed, then becomes a shared cor-

porate resource, the incentive for participation and contribution being ease of use.

Incorporation of Naming Standards

While naming standards for tables and columns in a RDBMS are essential to the data modeling process, the benefits of such standardization are often lost when translation of long column names to SAS variable names is performed. While most newer generation application development tools are capable of utilizing the long names provided for in RDBMS's, SAS code is very much limited to the eight character limit imposed by the Mainframe paradigm. While Release 7 will bring forth this much awaited capability, current systems development must cope with this limitation.

Given that when interfacing the SAS system to the RDBMS, there must be some translation from what is essentially a more descriptive name to one considerably less so, much variability and loss of information may result, particularly when development is done by programmers working quite independently of each other. If reliance is made on the automatic renaming features of the SAS/ACCESS product, even less intuitive names may result due to the rather simplistic algorithm employed which essentially truncates RDBMS column names.

A paper entitled 'SAS/ACCESS Techniques for Large DB2 Database'¹, presented at SESUG '93 by Craig Gavin, details several important techniques relating to automating the generation of code for SAS/ACCESS descriptors as well as PROC SQL Pass Through, however fails to address the SAS name mapping issues. To his credit, Mr. Gavin does touch upon several important techniques for improving the efficiencies of queries made against DB2, and provides several excellent techniques for generating appropriate 'WHERE' clauses based upon indexes. I hope to incorporate much of his technique into this system.

In the case of enterprise wide SAS application development, providing the middleware for preserving naming standards between database and programming language is the key to the development of re-usable code modules. By identifying the need for SAS naming standards in conjunction with the database naming standards, through customer involvement and various other means, we were able to get the data providers to provide support for this effort. In fact, while much of the metadata was derived from the data modeling and CASE tools, a method was developed for deriving SAS names from the DB2 and Sybase System 10 column

names. This technique is covered in depth in the Development Process section of this paper. To give you an idea of the magnitude of this undertaking, the FAST TRACK DB2 database contains over 3,700 columns in over 300 tables.

Addressing SAS Based Legacy Systems

Although the direction for much of the new application development effort has concentrated on tools which access data primarily from server platforms, until the distributed data warehouse is fully implemented, there exists a need for both maintaining mainframe based legacy systems as well as limited new development with portability and distributed processing capabilities built in.

In our shop, there exist many applications, running "in production" which require very minor maintenance, commonly referred to as "Legacy Systems", which more often than not, were written by independent end user programmers. There was sometimes a bit of code sharing, primarily through use of a public macro library, but even this was limited in scope.

Perhaps the most difficult issue to deal with is how to determine the need for programmers to modify existing application software to comply with newly established standards. Older code, particularly that written in the base language, often lacks the attributes which truly enable reusability. Furthermore, much code may have been written by an author who has "moved on" and is no longer accountable for a particular application. In such cases, there may be very little reward in retro-fitting these applications to comply with current standards.

When an existing source of data such as a flat file based on some extract ceases to be supported, the migration becomes mandatory. Input statements written against such entities become obsolete, and generally a view derived from PROC ACCESS or PROC SQL will become the vehicle for importing data into the SAS system. If left to their own devices, programmers will likely generate a multitude of views, none of which conform to any particular set of names, but rather are tailored to that specific program. This simply propagates bad habits.

For those applications for which there is no identified need to re-code or perceived benefit from recoding, a simple DATA step SET statement RENAME= option may suffice for modifying the code to conform to the new source of data. Providing a template wherein the new names are pre-defined and the user has but to type in the corresponding "old" names will facilitate this migration. In fact, as you will soon see,

this generation of a RENAME= statement is but one element of a larger code generation scheme ranging from pre-defined views to user-modifiable PROC ACCESS and PROC SQL modules.

It is opportunities such as this which form the basis of new development projects, while the underlying logic and coding sometimes is incorporated into the data at the database level. Even so, there is often an intermediate period wherein the existing application must continue to be supported regardless of the source of the data.

THE DEVELOPMENT PROCESS

First and foremost in the development process was the requirement for having a method of creating SAS names corresponding to both the RDBMS table names, to be used for naming the ACCESS descriptors and views, and for renaming column names to SAS variable names.

Originally an algorithm was written in SAS for parsing database specific names, based on identifying each functional node. Nodes were generally hierarchical and delimited with underscores. Pretty good results were achieved with key gauges of success being the resulting intuitiveness as well as uniqueness of the name, the former being a measure of usability. The latter, was considered a technical requirement based on uniqueness of column to SAS name mapping in the PROC ACCESS rename statement, as well as potential usage under conditions imposed by the joining of tables into views. This technique was good for a first pass, however a semi-manual update technique was finally adapted.

Technical Architecture Considerations - Accessing the Metadata from SAS

Initially designed as an application that could run on any desktop, the FAST TRACK metadata "browser" relied on a Microsoft ACCESS front end, with the metadata stored in an MDB format. This enables the programmer to toggle back and forth between the browser and the terminal emulation windows.

The data warehouse metadata browser relies on a series of Sybase System 10 tables and utilizes a PowerBuilder® runtime version front end.

The SAS/ACCESS interface to ODBC® provided as a beta product with SAS for Windows (R6.08, TS407) coupled with an ODBC driver for the

Microsoft Access® database, provided reasonable performance in extracting data from the FAST TRACK metabase. A view was created which joined two tables one containing table information, and the other, column information. This was in fact facilitated through the use of the SAS/ASSIST® Query window. Likewise, a PROC SQL view was developed utilizing the SAS/ACCESS interface to Sybase and Microsoft SQL Server®, also a beta product in release 6.08.

The structure of the respective metabase files was such that the table name, be it DB2 or Sybase System 10, the column names, and the associated SAS names were easily extracted into a SAS data set. This data set was then used as the single source of metadata information for all of the code generation routines. While this technique effectively freezes the data, this was not a concern for our purposes as metadata updates were very infrequent.

Technical Architecture Considerations - Storing the Generated SAS Code and Data Objects.

For the mainframe libraries, the choice was made to segregate Access Descriptors and Views into independent SAS libraries. Similarly, separate Partitioned Data Sets (PDS's) were defined to store the PROC ACCESS code for creating Views, and PROC SQL Pass Through code. This enabled using the SAS name assigned to the respective DB2 tables as the member name in the PDS. Each subject area was assigned its own set of PDS filenames with the abbreviation for the subject area used as a mid level node. The high level qualifiers defined the ACF2 access, while the low level qualifiers defined the type of library (i.e. SQL for PROC SQL Pass Through code, ACCESS.VIEW for the PROC ACCESS code, and in the case of the SAS libraries, ADLIB and VDLIB for the ACCESS Descriptors and Views respectively.

When generating the SAS code and members of the respective PDS's, a variable was assigned in such a fashion which concatenated the high level qualifiers as a character literal, the subject area variable from the metadata, the low level qualifier as a character literal, and the member name from the metadata. A FILE statement was coded such as to refer to this variable using the FILEVAR= option. The following is an example of using this technique:

```
if first.table_nm then do;
fn = "fasttrak."
    trim(left(subject))
    ".access.view("
    trim(left(tsas_nm))
    ")";
file prt filevar=fn;
...more sas code
end;
```

For the Sybase System 10 data warehouse SAS code libraries, a similar technique was applied, substituting directories on a shared file server as the repository in place of the PDS's and individual DOS files with an extension of SAS as the individual code modules for each table code.

```
if first.table_nm then do;
fn = "c:\fasttrak."
    trim(left(subject))
    "\.view\"
    trim(left(tsas_nm))
    ".SAS";
file prt filevar=fn;
...more sas code
end;
```

A full example of the code used to generate the code for the PROC ACCESS view creation as well as the resulting code module is provided as an appendix to this paper.

In addition to the 'external' file formats used for storing the SAS code modules, a technique was developed to move these elements into SAS Catalog based SOURCE entries. In the case of the FAST TRACK MVS PDS based modules, this was accomplished through the following means: Using the metadata SAS data set, a SAS routine was generated which opens the SAS Notepad editor, INCLUDE's the specified PDS Member which is simply the same as RDBMS table's SAS name, assigns a description label, and then saves as a like named SOURCE entry. The description was based on the underlying RDBMS table name for easy reference. Catalog entries are self documenting in terms of modification dates in the 'updated' date field in the Catalog list window (one must zoom the window to see these dates). This process repeated for each of the members of the PDS's.

The following example illustrates the code generated:

```
LIBNAME VDLIB
  'FASTTRAK.SUBJECT.VDLIB';

FILENAME VDCODE
  'FASTTRAK.SUBJECT.ACCESS.VIEW';

DM 'NOTEPAD;
DESCRIPTION "ANY DB2 TABLE_NAME";
INC VDCODE(anyDB2tb);
SAVE
VDLIB.SASCODE.anyDB2tb.SOURCE;
CLEAR;';
```

Note: Single quotes delimit the DM command.

The following code is used to generate the DM code required to save the source entries (Note: I use DB2TABLE as an generic example, this would be a real DB2 table name).

```
data _null_;
  set sasuser.tables;
  by subject;
  file 'fasttrak.sas(makesrc)';
  if first.subject then do;
    fname =
      "filename vcode 'FASTTRAK.'"
      || trim(left(subject))
      || ".ACCESS.VIEW;";
    libn =
      "libname vdlib 'FASTTRAK.'"
      || trim(left(subject))
      || ".VDLIB;";
  ;
  put libn / fname ;
end;
dmcmd1 =
  "DM 'NOTEPAD;"
  || "DESCRIPTION '"
  || trim(left(table_nm))
  || "';"
;
dmcmd2 =
  "INC VDCODE("
  || trim(left(tsas_nm))
  || "); SAVE VDLIB.SASCODE."
  || trim(left(tsas_nm))
  || ".SOURCE; CLEAR;'"
;
put dmcmd1
  dmcmd2;
run;
```

The entire process was written such that all subject areas were run at one time, with each defined as a unique catalog. SOURCE entries are stored in the same library as the respective descriptors and views, providing a certain level of convenience and documentation. For maintenance purposes, on an individual table basis, the

SOURCE entries provide a reasonable level of ease of use for the administrator. This method also provides an excellent example for developers to model their own libraries after.

Helpful Hint: To assist you in migrating SAS code from one platform to another, you can use a modified version of the PDSLIST macro provided in the BASE Sample Library to read the member names of any PDS into a SAS data set. Similarly, SAS code stored in a DOS files (with an extension of SAS) can be read into a SAS dataset by use of a DOS DIR command piped to a file which can then be read through a DATA step. Use the code, as given above, to copy the members of the PDS or DOS directory to Source entries in the catalog of your choice. You can then use SAS/CONNECT and PROC UPLOAD or DOWNLOAD, or alternatively create a SAS transport library, to migrate from one platform to another, moving entire libraries of SAS code within the catalog structures

Migration and Portability Considerations

Probably the most difficult issue to deal with in terms of implementing a data warehouse is the migration of production systems from one platform to another. Providing an intermediate source of data can ease the transition. This is truly the case when the model for the database is near to identical on both platforms although the actual RDBMS's may differ.

By instituting a repository available to legacy mainframe applications such as in DB2, with table structures, indexes, and table and column names identical to those in the distributed warehouse, the stage is set for easy migration. Even if physical data structures differ between platforms, transparency may be achieved through the use of views. Development of new applications as well as modification of existing applications can continue until the data warehouse can be installed and tested.

While the ACCESS descriptors and views will need to be regenerated to conform to the specifications of the new database, the code which interfaces with these descriptors and views need not change at all. This insulation from the underlying platform differences, is much like the "host layer" of the SAS Systems MultiVendor ArchitectureSM (MVASM). Through the process of loading the metadata into a SAS code generator for redefining descriptors and views, minor

modifications in the code generator are all that is needed to replicate the views from one platform to another.

End User Interface, Documentation, and Training

As detailed earlier under Technical Architecture Considerations, the SAS Access descriptors, views and code modules on the mainframe are made available through two storage mechanisms, and usage is a matter of programmer preference. Those more comfortable with the ISPF editor may wish to access the code in the PDS's while those working in the SAS System Display Manager may prefer working with the SOURCE catalog entries. Logical organization remains the same, i.e. grouping based on data subject area, be it a SAS catalog or PDS.

A metadata browser provides on screen access to current documentation and availability of corporate data. Regardless of the application, be it for the mainframe based FAST TRACK or the Data Warehouse, the browser runs as a Windows application and can be toggled or "hot keyed" to at anytime during the code retrieval process. The browser also provides cut and paste capabilities as implemented in the Microsoft Windows environment, enabling rapid and accurate lookups of specific code modules through SAS Catalog lists, ISPF member lists, and Windows File Open dialogs.

Documentation for the system has been automated wherever possible. SAS/ACCESS views are essentially self documenting through use of PROC CONTENTS, or through the DIR window. As a convenience, the contents of each descriptor and view are stored as unique entries in PDS's and DOS directories organized by subject area. The code modules include some information in a header area written as comment statements. As described earlier, the SOURCE entries are assigned a description corresponding to the associated RDBMS table name.

It is hoped that a minimal level of training would be required for usage of the elements of this system. As a general prerequisite to usage, we strongly recommend the SAS Institute training class suite of "SQL Processing with the SAS System" and "Processing Relational Databases with SAS/ACCESS Software". When offered in house, we generally work with the instructor to cover both DB2 and Sybase System 10 access techniques. DB2 and Sybase System 10 performance issues are covered to an extent in these courses, however we also provide internal classes specific to Sybase System 10 at the

developer and administrator level which provide much of the knowledge required for efficient usage of the data warehouse.

Futures

Aside from the enormous impact of migration from Version 6 to Version 7 of the SAS System, immediate concerns are the incorporation of solid Object Oriented attributes into all of the SAS code modules developed using these access methods.

At the time of this writing, plans are underway to develop a SAS/AF Frames based application for interfacing to the various code libraries and optionally, building queries from scratch. While we do not seek to provide the same functionality in terms of query building as found in the QUERY window of SAS/ASSIST, we do seek to provide some capability for retrieval and execution of the elements of this system. In particular, a method for generating subsets of the columns contained in any of the modules would be considered highly desirable, with an emphasis on "point and click" capabilities.

Further improvement to the system will be based on usage and efficiency concerns, particularly in the realm of query efficiency based upon incorporation of optional where clauses based on established indices. Logical joins and the like will also be provided, primarily based on the input from the end user programmers.

Conclusion

Through the application of well defined metadata incorporating standardized SAS names, we have produced an automated system for access to a corporate data warehouse based on distributed relational database architecture. Building on these concepts, we anticipate great success in creating a corporate asset in the form of a common software library of reusable SAS code modules.

Acknowledgments

Many thanks to the SAS/ACCESS product crew of SAS Institute, including Emily Wallace and Forrest Boozer, for listening to our pleas for a SAS for Windows Release based SAS/ACCESS interface to Sybase and Microsoft SQL Server® product and responding with a lot of special attention.

References

Gavin, Craig James, Jr. (1993). "SAS/ACCESS Techniques for Large DB2 Databases," Proceedings of the South East SAS Users Group First Annual Conference, pp. 99-105.

SAS, SAS/ACCESS, SAS/ASSIST, SAS/AF, SAS/CONNECT, MultiVendor Architecture, and MVA are registered trademarks or trademarks of SAS Institute Inc. in the USA and other countries. ® indicates USA registration.

DB2, MVS/ESA are registered trademarks of International Business Machines Corporation.

Other brand and product names are registered trademarks or trademarks of their respective companies.

APPENDIX A CODE USED TO GENERATE CODE

PROC ACCESS create descriptor code

```

filename prt 'c:\fasttrak\adcreate\creADVD.sas';
proc sort data = metadata;
  by subject table_nm;
data _null_;
  set metadata;
  by subject table_nm;
file prt;
if first.subject then do;
adlibnm = "LIBNAME ADLIB 'FASTTRAK.'"
  || trim(left(subject))
  || ".ADLIB";
vdlbnm = "LIBNAME VDLIB 'FASTTRAK.'"
  || trim(left(subject))
  || ".VDLIB";
PUT adlibnm;
PUT vdlbnm;
end;
ren = COLUMN_N || ' ' || CSAS_NM;
if first.table_nm then do;
  cre = 'CREATE ADLIB.' || trim(left(tsas_nm))
  || '.ACCESS';
  tbl = 'TABLE = DBA.'
  || 'TABLE_NM ' || ' ';
  ssid = 'SSID = 'DBXX';
  put 'PROC ACCESS DBMS=DB2;'
  / cre
  / ssid
  / tbl
  / 'RENAME'
  / ren
  ;
end;
else do;
  put ren;
end;
if last.table_nm then do;
  cre = 'CREATE VDLIB.' || trim(left(tsas_nm))
  || '.VIEW';
  put ' ';
  / cre
  / 'SELECT ALL;'
  / 'LIST ALL;'
  / 'RUN;'
  / 'QUIT;'
  ;
end;
run;
filename prt;
run;

```

PROC ACCESS user custom view code

```

data _null_;
set metadata;
by subject table_nm;
adlib = "LIBNAME ADLIB 'FASTTRAK.'"
  || trim(left(subject))
  || ".ADLIB' DISP=SHR;";
vdlb = "LIBNAME VDLIB 'your sas library file-
name;";
ren = COLUMN_N;
if first.table_nm then do;
fn = "fasttrak."
  || trim(left(subject))
  || ".access.view("
  || trim(left(tsas_nm)) || ")";
file prt filevar=fn;
cre = 'CREATE VDLIB.'
  || trim(left(tsas_nm))
  || '.VIEW';
ad = 'ADLIB.'
  || trim(left(tsas_nm))
  || ' ';
put adlib
  / vdlb
  // 'PROC ACCESS DBMS=DB2 AD=' AD
  // cre
  / @1 'SELECT'
  / @8 '/* DB2 Column'
  @48 ' SAS Name */'
  // @12 COLUMN_N $40.
  @48 '/* 'CSAS_NM $8. ' */'
  ;
end;
else do;
file prt filevar=fn;
put @12 COLUMN_N $40.
  @48 '/* 'CSAS_NM $8. ' */'
  ;
end;
if last.table_nm then do;
file prt filevar=fn;
put @8 ' ';
  // 'LIST VIEW;'
  / 'RUN;'
  / 'QUIT;'
  ;
end;
run;
filename prt;
run;

```

Appendix B Generated Code Samples

PROC ACCESS routine to create a descriptor with standard sas names

```
LIBNAME ADLIB 'FASTTRAK.SUBJECT.ADLIB';
LIBNAME VDLIB 'FASTTRAK.SUBJECT.VDLIB';
PROC ACCESS DBMS=DB2;
CREATE ADLIB.anydb2tb.ACCESS;
SSID = 'DBXX';
TABLE = DBA.any_db2_table_name;
RENAME
asof_dt      asofdt
base_nbr     basenbr
base_seq_nbr basesqnb
base_use     baseuse
;
CREATE VDLIB.anydb2tb.VIEW;
SELECT ALL;
LIST ALL;
RUN;
QUIT;
```

Note: This code is used only by the data warehouse administrator to establish descriptors for views to be generated from.

PROC ACCESS user module used to create custom view

```
LIBNAME ADLIB 'FASTTRAK.SUBJECT.ADLIB' DISP=SHR;
LIBNAME VDLIB 'your sas library filename';
PROC ACCESS DBMS=DB2 AD=ADLIB.anydb2tb;
CREATE VDLIB.anydb2tb.VIEW;
SELECT
    /* DB2 Column                SAS Name */
    asof_dt                      /* asofdt */
    base_nbr                     /* basenbr */
    base_seq_nbr                 /* basesqnb */
    base_use                     /* baseuse */
;
LIST VIEW;
```

Note: Commented SAS names are used as a ready reference. User edits this code to keep only those columns desired by deleting lines in this code.

PROC SQL Pass Through code: can be used as select, or to create a table or view

```
* LIBNAME VIEWLIB 'your SAS data library';
PROC SQL;
CONNECT TO DB2 (SSID=DBXX);
    /* CREATE TABLE SASUSER.anydb2tb AS */
    /* CREATE VIEW VIEWLIB.anydb2tb AS */
SELECT
    asofdt LABEL='ASOF_DT'          FORMAT=DATE7.0,
    basenbr LABEL='BASE_NBR'        FORMAT=11.0,
    basesqnb LABEL='BASE_SEQ_NBR'    FORMAT=11.0,
    baseuse LABEL='BASE_USE'         FORMAT=$78.0

FROM CONNECTION TO DB2
(SELECT
    asof_dt,
    base_nbr,
    base_seq_nbr,
    base_use
FROM DBA.any_db2_table_name)
AS T1(
    asofdt,
    basenbr,
    basesqnb,
    baseuse
);
QUIT;
```

Note: The labels and formats, derived from running PROC CONTENTS on the generic views, are provided as defaults which the user may override. The label is the corresponding table column name. User optionally uncomments CREATE statement.