Building a Purchasing Data Warehouse for SRM from Disparate Procurement Systems

Zeph Stemle, Qualex Consulting Services, Inc., Union, KY

ABSTRACT

SAS' Supplier Relationship Management (SRM) solution offers organizations the tools to analyze and manage purchasing information. This "spend" information is traditionally stored in procurement systems. SRM requires the data to be consolidated and summarized in a data warehouse. Therefore, the procurement information must be gathered from the various sources and housed in a common data warehouse.

Many organizations have multiple divisions resulting in multiple procurement systems. These systems often use different technologies, post the data at different times in a month, and store data in a variety of formats. Additionally, not all systems store the same information. For instance, one system may store purchase facts in U.S. Dollars and another in Euros. This presents an interesting challenge.

This paper discusses a solution for gathering procurement data from several disparate sources and storing it in a common data warehouse on a Win2000 server. The solution is fully automated and utilizes SAS Warehouse Administrator.

INTRODUCTION

SRM provides several tools for analyzing purchasing data and managing an organization's suppliers. It allows you to determine how much is purchased from any particular supplier or how much is spent on any particular item. You can also quickly see if there any disparities in how divisions purchase an item or if divisions are purchasing items from preferred vendors. The number of total suppliers can be reduced by combining purchases across division. Using this information, you can control and contain costs by consolidating purchases and leveraging your buying power with your suppliers.

For SRM to be used effectively the pertinent information from each the organization's divisions must be consolidated and summarized. This presents an interesting challenge. How can we gather this information from several disparate systems in a timely fashion and how can we consolidate it so that it is useful with the SRM analysis tools? Remember, any system is only as good as the data with which it is provided.

The following methodology is one way to meet this challenge. This paper provides a high-level overview of this and gives simple examples to help you understand one approach. The following tasks are discussed:

- Developing a data model to provide a structure for the data.
- Having each division supply data in delimited flat file format using the guidelines in the data model.
- Building a data warehouse to store and consolidate the data.
- Developing an application to read the flat files when they are available and populate the data warehouse.

SAS SUPPLIER RELATIONSHIP MANAGEMENT

The SRM solution contains several tools to help a purchasing organization analyze their spend. The following is a list of these tools and a brief description of each of them:

- Spend Analysis – A Web-based HOLAP tool that allows you to view purchasing data through hierarchies and customizable views. Spend Analysis allows to user to subset data, create custom views, and export data.
- Scorecard – A graphical dashboard that displays how well an organization is meeting preset goals.
- Ranking – A tool that ranks suppliers based on preset criteria so an organization can easily identify top suppliers.
- Reports – A series of Web-based reports that provide a quick look at purchasing data.

As you can see, the ways purchasing data can be presented and analyzed are practically limitless with this suite of tools. However, the tools and their uses are only as good as the underlying data. If you provide bad or incomplete data to SRM, the tools will reflect that. The secret to harvesting savings from SRM is to warehouse quality, clean, complete data.

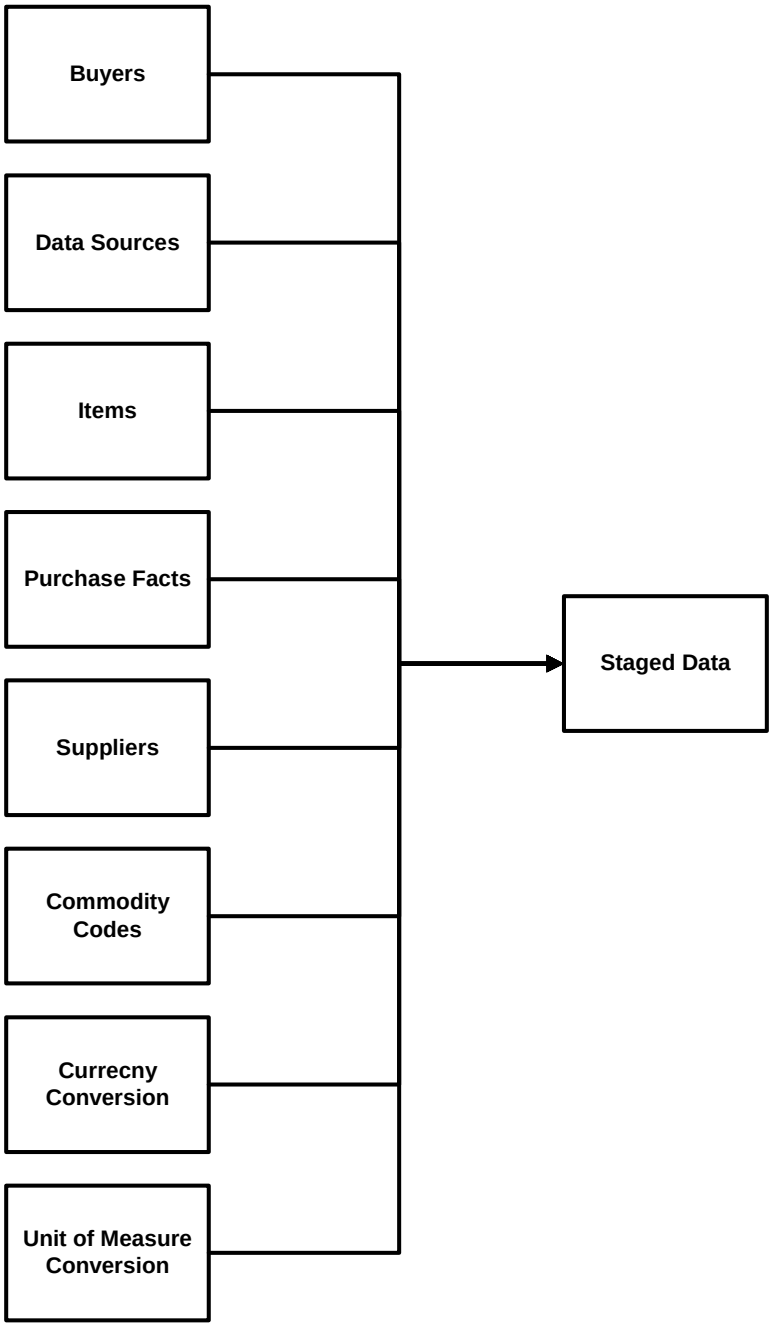
DATA MODEL

The objective when you develop a data model is to design it so as much purchasing and supplier information is included as is possible. One important consideration when designing the model is to include representatives from each division in the discussions. Use people that understand their data and can provide meaningful input to the model. This helps you get buy-in from the divisions and also helps flush out any issues early on.

Chances are, not all data will be available from all divisions. For example, one division may include payment dates and another may not have the data for payment date. Considerations must be made in the data model to compensate for differences in the data. A good rule of thumb is to design the model to include all possible fields from all divisions.

While it is important to include as many fields as possible in your data model, you can go too far. If your data model is too complex, it will difficult for the divisions to provide the necessary data to meet it. If the vast majority of the divisions cannot provide data for a specific field, it may be better to not include that field at all. Of course, ensure every effort is made to extract information that is the most useful. Decide as a group how useful the data is to all parties and the use that information to decide whether or not to include a field.

The following diagram shows a simple data model. Your organization may need more data entities to best represent your information:



Each of the blocks on the left represent an entity in the data model and a corresponding delimited flat file that is provided by each division. Each of the entities must be able to be joined through the use of key fields. Each entity must include one or more keys that enable all the data to be consolidated into a Staged Data entity.

The Purchase Facts file is the centerpiece of the data model. It should contain line item detail for all purchases. It also includes the appropriate key fields necessary to link each record to the matching members in each supporting table. For example, each record contains a Buyer ID that points to buyer detail in the Buyers entity, a Supplier ID that points to supplier detail in the Suppliers entity, and so on. It may be necessary to add additional fields such as a Plant Code or a Source System ID to each key to ensure it is unique. This becomes necessary when multiple divisions use the same codes to represent different objects (e.g., buyers, items).

The following table provides a brief description of each entity and the type of information it contains:

Data Entity	Description	Keys
Buyers	List of buyers including any organizational hierarchies	Buyer ID, Source System ID
Data Sources	Identifiers for data sources	Source System ID
Items	Part numbers including hierarchical descriptions	Part Number, Commodity ID, Source System ID
Purchase Facts	Line item detail for purchases	Buyer ID, Source System ID, Part Number, Supplier ID
Suppliers	Vendor information including addresses	Supplier ID, Source System ID
Commodity Codes	Enrichment data to standardize items	Part Number, Commodity ID
Currency Conversion	Look-up table for currencies	N/A
Unit of Measure Conversion	Formulas for converting item purchases into standard units of measure	Part Number

In addition to specifying what data entities are going to be needed for SRM, the data model should also document what data columns are going to be included in each entity and the attributes for each column. While these specifications are likely to change somewhat as the project proceeds, the closer you get to the final design up front, the better off you will be. Adding columns and changing column attributes can be tricky after you begin populating the data warehouse. Additionally, avoid changes requested by one division that may adversely affect another division. Design with the overall good of the corporation is mind.

You should include as much information as possible about each column. Attributes that definitely should be included in the data model are name, length, format, and informat. Optional attributes can be label, valid values, and any other pertinent information. Remember, the better you define data before building the data warehouse, the easier the warehouse will be to build and maintain.

It may also be necessary to freeze the data model at some point in time during development. If you are not careful, the stability of the data model can be jeopardized by constant change. Agree on a scheduled date to freeze the model and then adhere to it. Of course, emergency changes may still be necessary.

PURCHASING INFORMATION

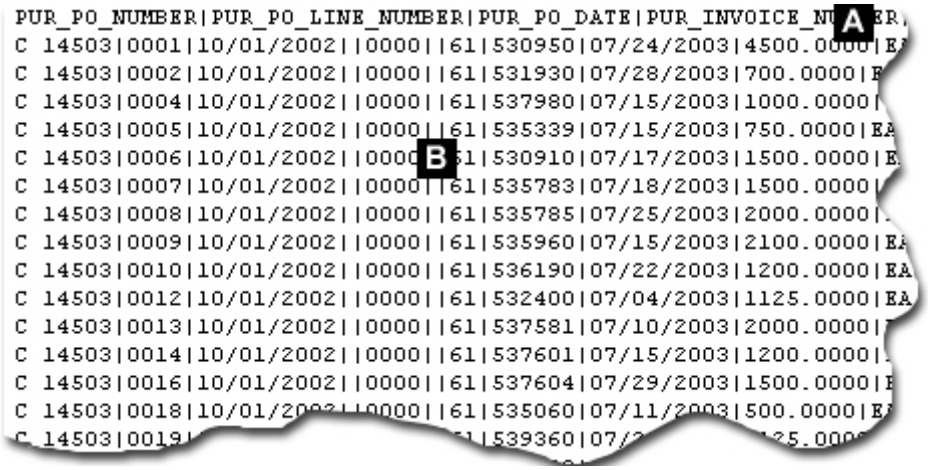
Once the data model is built and agreed upon by all divisions, each division can begin to export purchasing information to a common directory. In our example, the divisions produce delimited flat files that are copied to an ftp site on a monthly basis. Flat files are relatively small and can be easily produced by most systems. Each division can be assigned a folder on the ftp site where they can post their files. Each will produce the same set of files each month (e.g., Buyers, Purchase Facts).

Initially, we used comma separated files for the purchasing information. However, this presented issues due to commas embedded in description fields. The only way to produce csv files that include embedded commas is to enclose all fields in double-quotation marks. This was not easily accomplished by the divisional programmers developing the purchasing data export programs. Therefore, the final solution was to use the vertical bar symbol (|) as the delimiter because no vertical bars were included in any of the data.

The first set of data provided by each division normally includes several months of data. After the initial load, only new information is provided on a regular basis. This “data refresh” is appended to the existing data. If no new information is available for a particular data entity, blank files that only include a header record are produced. Avoid sending unnecessary data in these refresh files as the size of the data warehouse can be severely impacted.

In an ideal world, each purchasing system would be identical with each containing the same fields of information. However, most of us do not live in such a world and a more flexible alternative is normally required. Divisions that do not have information to populate each field in the data model do not have to provide that missing information. The application that reads the file (discussed later) reads only the included information and populates the data warehouse with blanks for the missing fields. This allows each division to provide different information while maintaining a consistent set of tables in the data warehouse.

The following is a snapshot of a Purchasing Facts delimited file:



The first line of the delimited flat file contains the field names for the fields included in the file **A**. This is called the header record. It is very important that these names are used consistently across all divisions. The data model specifies the names to be used for columns. If the system is to be automated and streamlined, these names should be adhered to by all. If a division does not have data to populate a particular column, they can simply leave it out of the file. For fields with missing information, the delimiter is used as a placeholder. **B**

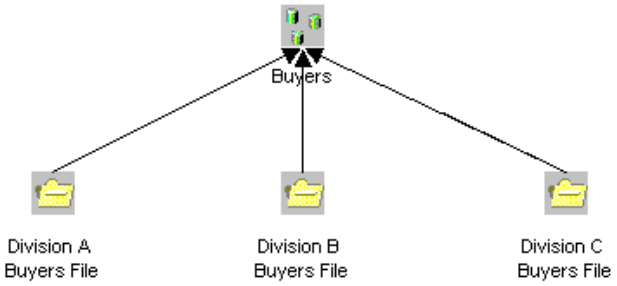
Each record in the delimited file needs to have exactly the same number of columns. Additionally, the header record must have the same number of columns as each data record.

DATA WAREHOUSE DESIGN

SAS' Warehouse Administrator product enables you to build, schedule jobs, document, and administer a data warehouse that can be used in a variety of applications. The SRM product requires that you use Warehouse Administrator for the staged and summarized data used in its reports. SRM and Warehouse Administrator work well together allowing you to build a robust application that can be easily maintained. You can also include your own code for functions not specifically covered in Warehouse Administrator.

The data warehouse used in our example follows the data model exactly. There is an Operational Data Definition (ODD) for each type of delimited flat file. These ODDs are used to provide access to the delimited flat files. An ODD is a metadata record that specifies the physical location of each file and how each file is to be loaded. The load step for the ODD is attached to the application's source code that actually reads the file. The ODD also contains information about each column of data, such as length and format.

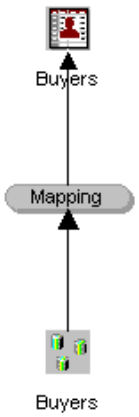
The following diagram shows an example of an ODD process:



A corresponding Data Table is created in the data warehouse for each ODD. A Data Table is a metadata record that that specifies an actual SAS table or view. In our example, a Data Table exists for each table in the data model. This one-to-one relationship provides a smooth throughput to get the data from the flat files into the data warehouse tables.

Data is appended to the appropriate Data Table as it is read from each of the delimited flat files. The Data Table is mapped to the ODD using a 1-to-1 mapping technique. This means that the columns and their attributes are exactly the same in the Data Table and the corresponding ODD. The Data Tables are eventually used to build the Staged table and the NWAY table.

The following diagram illustrates the relationship between an ODD and a Data Table:



Finally, each Data Table is merged with the other Data Tables to form a staged tabled which is then summarized to build an NWAY table. The summarized information is used in the SRM reporting processes and the staged information is available for reach-thru.

APPLICATION TO READ PURCHASING FILES

One of the biggest challenges in any data warehouse is how to acquire the necessary data from the various data sources. Normally this data comes from a variety of disparate sources and is available at different times. A solution is needed that will consider these challenges and provide a resolution that is easy to maintain.

As stated earlier, delimited flat files work nicely as a means to get information out of the purchasing systems and into the data warehouse. Delimited flat files are:

- Relatively easy to create from most systems.
- Significantly smaller than structured data files and therefore easier to transfer.
- Easily read by SAS program code.

In our example, each division is provided a folder on an ftp site in which to place their files on a monthly basis. Initially, two years worth of data was transferred to the warehouse. After that, only information that is new is placed onto the ftp site. If there is no new information for a particular data entity (e.g., no new buyers for a given month), a file containing only the header record is transferred.

PROC IMPORT VS. DATA STEP

While PROC IMPORT can be used to read delimited flat files, this solution did not work well in our example. PROC IMPORT reads the first several records to determine whether a column is numeric or character. If a character appears in a field that is assumed to be numeric, errors occur. Additionally, truncation can occur if a column’s length is not properly determined. Based on these limitations, a DATA step was required to read the flat files.

READING THE FILES

The DATA step written for this solution had to be somewhat dynamic. Not all divisions are supplying the same columns of information. Therefore, the INPUT statements that read the information must be created “on-the-fly” based on what is in each file. This was accomplished through the use of parameter datasets for each data entity and a macro to read the files. This technique allows you to use a single piece of code to read all of the delimited files.

A parameter dataset was developed for each data entity using the information in the data model. This dataset is used to control the DATA step that reads the files. Each parameter dataset contains information about every column that may be in each file. The following information is included in each parameter dataset:

- Variable name
- Length
- Informat
- Format
- Label (optional)

A single macro can be used to read all the delimited flat files. The name of the file being read is passed to the macro causing it to read the variable information from the appropriate parameter dataset. The first line of the file is also read to determine which columns of data are available. This is merged with information from the parameter dataset.

A DATA _NULL_ step is used to create macro variables for the INPUT statement. Any columns that are available will be read from the file. Null values will be generated for any columns not available. This is to maintain consistent data regardless of which columns are available. This also prevents errors and warnings when the data is appended to the warehouse tables. The number of columns is determined and then used as a control for the DATA step that reads the file. A flag is set for each variable that indicates whether or not it is available in the flat file.

The code also has the built-in ability to detect which columns are present in the header record and which are not. For the columns not present, the program creates the variable and populates the value with missing for numbers and null for characters. This technique forces the dataset to have the exact same structure regardless of which columns are present in the input file. This is also necessary to prevent any errors when appending the new data to the existing data in the data warehouse. It also allows you to accept data from a variety of sources even if the same data is not available from each of them.

The following is a sample of the DATA STEP included in a macro used to read a delimited flat file. Keep in mind that this is only a small piece of the entire application.

```
data readfile;
 infile &fileref delimiter = "&delim"
        missover dsd recfm=V lrecl=37726
        firstobs=2;

 /* Length statement */
 length
 %do l = 1 %to &num;
   &&varname&l &&length&l
 %end;
 num 8 rep 8;

 /* Informat statements */
 %do x = 1 %to &num;
   %if &&present&x = 1 %then
   %do;
     informat &&varname&x &&informat&x;
   %end;
 %end;
```

```
/* Format statements */
%do y = 1 %to &num;
  %if &&present&y = 1 %then
  %do;
    format &&varname&y &&format&y;
  %end;
%end;

/* Input statement */
input
%do z = 1 %to &num;
  %if &&present&z = 1 %then
  %do;
    &&varname&z &&type&z
  %end;
%end;
;

/* Null values for missing columns */
%do zz = 1 %to &num;
  %if &&present&zz = 0 %then
  %do;
    %if &&type&zz = $ %then
    %do; /* Character variables */
      &&varname&zz = ' ';
    %end;
    %else
    %do; /* Numeric variables */
      &&varname&zz = .;
    %end;
  %end;
%end;
```

Once the data is read, it is placed in a temporary table and then appended to the appropriate Data Table in the data warehouse. The temporary table is then deleted.

DETERMINING WHEN DATA IS AVAILABLE

There are several ways to control when data is read. A solution is needed that can determine if the files on the ftp site are new and then only read them if the information they contain is not yet in the data warehouse. If you read a file that has already been processed, duplicate entries will be appended to the warehouse.

The solution was to store the date each file was last processed. These dates are then compared with the file dates on the ftp sites. Any files that are new are processed and then the stored date is updated. This allows new data to be processed and added to the warehouse as soon as it is available and also prevents reading the same data twice.

CONCLUSION

Successful design, development, and implementation of a data warehouse is paramount to the success of an SRM application. Including all divisions from the beginning is one of the enabling factors for the success of your warehouse. Design with everyone in mind and you will end up with a data warehouse that provides data the affects the bottom line.

CONTACT INFORMATION

Your comments and questions are valued and encouraged. Contact the author at:

Zeph Stemle
Qualex Consulting Services, Inc.
10285 Hempsteade Dr.
Union, KY 41091
Phone: 859-384-3072
Fax: 603-949-7880
Email: zeph.stemle@qlx.com
Web: http://www.qlx.com