ABSTRACT

Building a data warehouse is a complex process involving many different phases; everything from database design, to hardware configuration and benchmarking, to user documentation and training. The authors of this paper have successfully implemented two large (20 GB) data warehouses using SAS on a Unix platform and will use their project as a case study to examine the process of building a data warehouse, from inception to delivery. The major phases of the process (analysis, evaluation, design, and implementation) will be discussed in detail, examining various issues and strategies. Special attention will be given to the topic of metadata and the key role it plays in the implementation of a data warehouse.

INTRODUCTION

Data warehousing has been a hot topic for the last couple of years. Today, there are conferences and institutes devoted exclusively to this one topic. Why the interest all of a sudden? First, it is really not all that sudden. The creation of extracts from operational databases, the storing of these extracts in a separate location (maybe even another platform), and the use of these extracts for decision support has been going on for years. Well then, why all this attention now? Primarily because the successful implementation of a data warehouse holds the promise of many benefits that can positively impact an organization's bottom line and help them to remain competitive - benefits such as the availability of an integrated, consistent, easily accessible source of data, designed and optimized specifically for information delivery. Also, thanks to recent advances in client/server technology and the proliferation of end user tools for data access, manipulation, and presentation, it has become possible to empower business analysts and knowledge workers by giving them tools to directly access the data warehouse and perform their own analyses.

Since the benefits of building a data warehouse can be significant to an organization, it is important to those thinking of implementing one to understand some of the factors that can help to ensure success. The authors of this paper have spent most of the last three years implementing, enhancing and maintaining two large data warehouses on the Unix platform using the SAS System. The intent of this paper is to share some of the factors we feel have helped our project succeed.

Building a data warehouse is a complex process involving many phases. At a very high level, one can look at this process from two different, but complementary, perspectives: the business perspective and the technical perspective. These two perspectives affect, to varying degrees, each of the major phases involved in building a data warehouse: analysis, evaluation, design, and implementation. This paper will discuss each of these phases in turn, examining the business and technical issues relevant to each. It is important to always keep in mind that building a data warehouse is not simply an exercise in technology - ideally, it should be an activity whose primary drivers are the needs of the business community. A key factor in the success of our implementation is that, in addition to our technical abilities, we have strong roots in understanding and supporting the needs of our customers, both inside and outside our organization.

ANALYSIS

As with any software engineering project, building a data warehouse begins with analysis. In this case, the end result of the analysis process is the construction of a data model that will describe the logical design and data elements that comprise the warehouse. The first step in the analysis process is to determine who your customers are - that is, who will be the users of the warehouse. In our case, this was an important step, since we support a wide
variety of both internal and external customers and these customers represent a wide variety of data requirements. One conclusion we reached very early on (and one that became a theme throughout the entire implementation), is that we had to start small, set realistic goals, and remain focused on these goals throughout. It was not realistic to expect that the data warehouse would meet all the needs of all our customers. So part of the analysis process was to determine which customers, and consequently which applications, would be candidates for migration to the alternate platform.

In order to make an intelligent decision, we established some criteria by which we would judge candidate applications. This helped us to approach the decision-making process in a logical and organized manner. In our case, we established three criteria: (1) An application had to be supported by a well-defined, reasonably-sized set of data elements. We have terabytes of information on the mainframe. Clearly, we would be able to move only a fraction of this to our warehouse. (2) The application had to consume a significant portion of resources - both staff and mainframe. In this way, we would maximize the reduction of mainframe cycles as well as the associated costs. (3) The application should be tied directly to revenue. We believed that we could significantly shorten both the development and fulfillment cycles with our data warehouse and thus produce our products in a more timely and efficient manner. It was important to the current and future success of our project that we pass this benefit on to our revenue customers. The importance of associating revenue, either direct or indirect (e.g. through reduction of mainframe costs) cannot be overstated. By tying a revenue stream to the data warehouse, it becomes much easier in the future to justify the cost to upgrade and extend the warehouse.

Once we had chosen the applications that our data warehouse would support, we then performed an extensive analysis to determine the data elements that were most frequently requested and/or utilized by those applications. This analysis gave us the data model for our warehouse. We then reviewed this model with our customers to make sure that their requirements were satisfied. It is critical to get feedback from the appropriate players during this phase of analysis.

**EVALUATION**

Concurrently with the analysis phase, we began the evaluation phase. Here the purpose was to evaluate and select both the hardware and software for our alternate platform. As in the analysis phase, the first thing we did was to establish some criteria to guide our evaluation. The major criteria we identified for evaluating software for our warehouse were: (1) The time it would take to load our data warehouse, (2) The time and effort it would take to implement the data warehouse and transition our group to a new development environment, (3) The availability of an integrated set of software tools for data management, analysis, and presentation and (4) Portability and scalability. Because technology is rapidly evolving, a key element in the long life of any solution is that it is both portable and scalable. Portability leverages the existing development effort and significantly reduces the need for re-engineering applications simply because they are moved to another platform or environment. Scalability ensures that you can start small and grow your application incrementally, as the business demands.

Armed with these selection criteria, we next identified a group of vendors to evaluate. Since our company had formed a strategic alliance with Hewlett-Packard®, the choice of hardware platform was made for us. In the software area, we chose to evaluate Sybase®, Informix®, and SAS as potential data management systems for our data warehouse. Sybase and Informix had already been used in alternate platform implementations within our organization, and Sybase, in fact, was the de facto standard. We chose to evaluate SAS because of our extensive experience with the product.

The actual evaluation process consisted of meetings with the vendors, demonstrations of their products and supporting tool sets and finally, benchmarking.

**Benchmarking Methodology**

As with most things in life, a planned approach often yields the best results. In the area of benchmarking, taking an organized and methodical approach is required, if the results are to be meaningful. In building our data warehouse, we had to perform benchmarks to assist in the configuration of both our hardware and software.

Even though HP had already been identified as our hardware vendor, we still had to devote significant time and effort to determine what type of machine and disk would best suit our needs, and how they should be configured for optimal performance. Our relationship with HP was very important here, since they were willing to give us hardware on consignment that was very similar to that which we would be using in production, as well as making a
Information Systems

system engineer available to help us with testing and configuration. Some of the factors we examined during our testing were: the number and type of disk devices and disk arrays, the block size in which the disks were formatted, and the RAID level in which the disk arrays were configured. RAID offers protection of the data in case of disk failure. RAID level 3 is best for applications that both read and write. However, as the number of concurrent processes increase, disk performance suffers. RAID level 5 promotes concurrent disk access and is the preferred format for read-only applications. RAID level 3 would give us a faster performance when we had multiple concurrent users extracting data from the warehouse.

In addition to testing and tuning hardware, we also tested the software for our data warehouse. When we began to examine the SAS System as the DBMS for our data warehouse, we first had to decide whether we should create SAS datasets on the mainframe, after our data was extracted, or download "flat" files and then create SAS datasets on the Unix platform. One guiding factor in all our benchmarking was that we wanted to be as efficient as possible in our use of resources (e.g. CPU time, disk space, the number of tape drives required). Due to a mainframe downsizing initiative, it became increasingly difficult to get mainframe resources. Extracting and processing 20 GB of data (the size of our data warehouse) takes a tremendous amount of resources. It was decided that we would utilize our resources more efficiently by downloading ASCII "flat" files to the Unix platform and running SAS there to create our database tables.

Since there is a limitation in the Unix operating system that no single file can be larger than 2 GB, we have to segment our data on the mainframe, prior to downloading, into files that are less than 2 GB each. In order to determine the optimal number of files that could be downloaded simultaneously, we benchmarked by downloading 1 file, 2 files, 4 files, and 8 files concurrently, through both a token ring connection and an FDDI (fiber optics) connection. We collected statistics on throughput (in KB/sec) and determined that optimal throughput was achieved with 4 concurrent FDDI downloads.

We next moved on to the database loading process. Again, since we had multiple files to load, we wanted to determine the optimal level of concurrency. We tested loading 1 file, 2 files, 4 files, and 8 files concurrently, trying to partition the jobs evenly among 4 disk controllers. As before, we collected statistics on load time (in seconds of CPU time and "wall" time) and determined that optimal throughput was achieved with 4 concurrent SAS processes to create and index our database tables.

The final area we benchmarked was the extraction of data from the warehouse. In general, there are two ways to extract data: (1) sequentially, processing all or some of the database records, applying selection criteria to either include or exclude any given record, and (2) randomly, using a primary key to extract specific records. Since the applications we chose to migrate used random access about 75% of the time, it made sense to try to optimize this operation. For both types of access we looked at a variety of factors and examined how each affected both CPU time and "wall" time. These factors were: the size of the record (in bytes) being extracted, the number of concurrent processes, the use of one vs. multiple SAS work areas, and the RAID level (3 vs. 5). In addition, for random access, we examined the effect of varying the number of keys used for indexed retrieval and, for sequential access, we examined the effect of varying the number of records selected for output.

At the time, implementing alternate platform systems was still very new within our organization and many issues were being addressed for the first time with our project. In addition to the tests mentioned above, we had to deal with network configuration issues and spent several months finding the right configuration for our PCs to access the Unix platform.

In the final analysis, we chose the SAS System for our data warehouse for its ability to quickly load the database (10 hours vs. 3-7 days for relational databases), for its extensive product line which gives us tools to store, extract, manipulate, analyze and present our data, for its powerful client/server capabilities, and because it allowed us to leverage our core competency and quickly implement our data warehouse. Because SAS language is virtually platform independent, our developers' SAS coding skills helped to make them productive immediately, with very little retraining necessary. Even though SAS was an unconventional choice, we had the support of our management. The benchmarking we performed was a key factor in proving that SAS was a viable alternative to more traditional relational databases.

DESIGN

The hardware platform and the database management system for the warehouse have been chosen. The users of the warehouse have been identified and a data model that meets their
requirements has been constructed. The next step is to determine the physical design of the warehouse, as well as the mechanisms by which users will access and retrieve data.

Segmenting the Data

As previously mentioned, there is a 2 GB file size limit in Unix that SAS has to honor. So the first design issue to address, if you have more than 2 GB of data, is how to segment the data. Given that everyone has different data and different data retrieval requirements, there are two general approaches, described below, that one can consider.

For the first approach, if you have one or more data elements in your data warehouse that comprise a primary key, you can simply sort your data by the primary key and segment the data sequentially by this key into data sets that are 2 GB or less. You can then create a table (or perhaps a set of macro variables) which tracks the minimum (or maximum) value of the primary key for each data set. Data retrieval is then straightforward. If you wish to retrieve data using the primary key, you would first match your keys to the minimum (or maximum) values in each data set to determine where each key was stored and then go directly to the appropriate data set to extract the data.

The second approach is a bit more complicated to construct and support, but the payoff is a potentially significant savings in processing time. First, ask yourself these two questions: Are there certain subsets of data that I process all the time? Are there other large subsets of data that I frequently want to exclude from processing? If your answer to these questions is yes, then your warehouse is a candidate for the following approach: Identify those variables that subset your data into groups that are frequently accessed versus those that are less frequently accessed. Combinations of the values of these variables will serve to create mutually exclusive categories into which each record in the database can be placed. For example, let's say you have identified three variables, A, B, and C, that satisfy the above requirements. Let's also assume that each of these variables is binary, having only two possible values. There are then $2^{3} = 8$ possible combinations of values of these three variables that can be used to segment the data. All 8 combinations need not be used - some can be combined if they are not required for subsetting.

The payoff with this approach comes from the fact that you can automatically (and easily) exclude large subsets of data, simply by not reading the appropriate data sets, saving I/O's and CPU cycles.

Of course, the downside is that you must be able to segment your data according to the values of these key variables - not an easy task if you are processing gigabytes of data. In our case, we have two data warehouses: one uses the first approach, segmenting the database sequentially by a primary key. The other uses the second approach. In this case, we have identified five key variables to create our segments. These variables generate 48 different combinations of values. However, we have reduced this to 13 combinations that are meaningful in our applications. On the mainframe, we have programs which interrogate the values of these key variables and map them into the 13 categories. Another program then creates the database segments based on this mapping.

One additional solution to data segmentation is to denormalize the data by separating out groups of related data elements and storing them in their own data sets. This make sense if there are groups of related data elements that are sparsely populated but occupy a large amount of space. Creating a separate data set for each group of elements can significantly reduce the total amount of space occupied by the warehouse as well as significantly reduce the time required to build the database. Again, there is a downside as well - more I/O's are required to retrieve these data elements since they are stored in separate tables.

Indexing the Data

Another element of database design involves deciding what fields should be indexed and what type of index structures should be created. As with most things, there are tradeoffs involved with indexing. Indexing can significantly improve data access and retrieval times. The downside is that, of course, there is overhead involved in creating indexes - overhead in terms of the CPU time required to construct the index structures and overhead in terms of the disk space required to store the indexes. Analysis is required to determine what data elements are appropriate for the creation of indexes. You need to look at what variables are frequently used as selection criteria and how many different values those variables can take on. Variables with only a few values (e.g. sex) are not good candidates for indexing since there are many records associated with each value of the index. Variables which take on many discrete values are better suited to indexing.

The SAS System does support the creation of indexes. Indexes can be simple, involving just one variable, or they can be composite, involving two or more variables. An index can also be specified as
unique, where each value of the index occurs only once. Physically, indexes in SAS are stored separately from the data, although SAS views them logically as one entity. This is beneficial in the Unix environment, since the additional space occupied by the index does not impact the 2 GB limit for the data. That is, the data set can occupy up to 2 GB and the index can also occupy up to an additional 2 GB. However, the data set and the index must occupy the same physical directory. They cannot be stored on different file systems.

Depending on how you segment your data, you may have to get creative in building your index structures in order to facilitate data retrieval. The approach described above, of segmenting the data based on key variables, means that, unless the primary key is one of the variables used to segment the data, there is no efficient way of using the primary key of an index to retrieve data. One approach (the one we took) is to create an additional data structure as a first-level index - one that contains one record for each primary key value plus some sort of pointer that indicates in which data set that primary key is stored. This data structure should also be indexed on the primary key. Data retrieval then becomes a simple two-step process - first, match your keys to the first level index and retrieve the pointers and second, follow those pointers to the appropriate segment. In the process of loading our data warehouse, this first level index is created after all the data tables have been built. We simply set together all the data sets and create the first-level index structure by keeping only the primary key and a derived code that maps each key to its data set.

### Metadata - A Key Success Factor

One of the major success factors in any data warehouse is ease of access to the data in the warehouse. The need for relatively complex segmentation and indexing schemes makes data access and retrieval more difficult. In order to ease the burden on the user, a data access layer is needed. This layer serves as the interface to the data warehouse and makes access to the underlying database structures easy, and changes to these structures transparent to the end user. Metadata is the driving force behind this data access layer.

Metadata, simply put, is data that describes the data in the warehouse and its creation is probably the most important aspect of building a data warehouse. Metadata plays a key role in the architecture of the data warehouse because it helps users (applications, as well as humans) understand and navigate the data warehouse.

The SAS System does make some metadata available. The way to access this metadata is by using PROC SQL to interrogate the DICTIONARY tables. These tables have been available since release 6.07 and give you access to metadata relating to SAS catalogs, SAS data sets (as well as their variables and indexes), SAS libraries (those associated with a file), and external files (those associated with a fileref), among other things.

Although there are many ways to organize and structure metadata in the SAS environment, we propose that there are three major categories of metadata that need to be collected:

1. **File System Metadata.** This describes the file systems (i.e. directories) available for use by the data warehouse and includes the name of the file system, its size (in MB), the types of data that will be stored there (e.g. flat files needed to load the warehouse, the SAS data sets that comprise the warehouse, or SAS work space), and the percentage of space available for each type of data the file system will hold.

2. **Data Set Metadata.** This describes the SAS data sets that make up the data warehouse and includes the name of the data set, a description of its contents, the file system on which it is stored, whether it is indexed and on what fields, and how often it is loaded (i.e. monthly, quarterly, on demand, etc.).

3. **Variable Metadata.** This describes the data elements that make up the data warehouse and includes the variable name, its type (numeric or character), its SAS length, any associated informat or format, its mainframe source, a long description of its contents, and in what data sets it is stored. Some of this information is available in the DICTIONARY tables referenced above. The rest has to be created by other manual means. All metadata can be stored in SAS data sets.

Utilizing this metadata, we have been able to construct a series of SAS macros that encapsulate the intelligence required to extract information from the warehouse. From the user's perspective, they simply need to invoke a macro and provide the list of data elements they wish to extract, along with any selection criteria, and the rest is taken care of for them. Changes, such as the addition of new segments, the movement of segments from one file system to another, and the deletion of segments, are all transparent to the user. Additionally, this metadata serves as documentation for the warehouse that can be made available to the user either in hardcopy form or through a GUI interface.
We are currently working on significantly expanding our use of metadata by utilizing it to build a database administration application. This application will provide a GUI front end to manage the creation and modification of metadata and will allow users of the data warehouse to query the metadata. This application will also use the metadata to automate the process of loading the database, so that when new segments are added, or segments needed to be moved to another file system due to an increase in size, the scripts that control the loading process will be automatically modified to reflect the appropriate changes.

IMPLEMENTATION

Almost a year transpired from the start of the project to the time when we were ready to actually begin using our data warehouse for production processes. As with other phases of our implementation, we started small. Initially, only the developers in our own workgroup were targeted as users of the warehouse. In order to maximize our chances of success and to prove the viability and usefulness of the data warehouse, the concentration was initially on those users who needed the least amount of training and who could be productive the fastest.

The next phase was to conduct training sessions for some of our more technical business users. Most of these users have had some exposure to SAS. We conducted a full-day training session that covered not only the data warehouse, its contents, and how to use it, but also included a basic introduction to the Unix operating system. These users have been given direct access to the warehouse and have been using it quite successfully. The final phase of the rollout is targeting the non-technical end user. To meet the needs of this type of user, a GUI PC front-end is being developed that will access the Unix data warehouse. Users will be able to extract data simply by pointing and clicking, without having to write any code.

A few words are in order, at this point, regarding the Unix environment and the challenges it presents to those who would like to migrate processing off the mainframe. As users of the mainframe, we are used to having resources managed for us transparently. Job scheduling and load balancing take place automatically. When files on disk have not been referenced for a certain period of time, they are automatically migrated to tape. We can access files on tape and can often arrange to get substantial quantities of dedicated DASD when the need arises. Not so in the Unix environment. The Unix operating system, "as is", does not come with software that will automatically queue up jobs if too many are submitted at the same time. Unless scripts are written to compress, backup, and/or delete files, you will very quickly find yourself with file systems that are 100% full and files that, once deleted, cannot be recovered. The price you pay for having a dedicated platform is increased responsibility - responsibility for intelligently managing all the resources on that platform. It is wise, from the beginning, to put standard procedures in place to monitor and manage resource utilization, as well as to educate users on how to become responsible Unix citizens. The good news is that more and more products, similar to those found on the mainframe, are becoming available to help manage Unix resources. The bad news is that first, they must be evaluated and tested before they are placed into a production environment and that takes time and effort.

CONCLUSIONS

In summary, the key factors that we believe can help to make the implementation of a data warehouse successful, are the following:

(1) Technology should not be the driver - keep the business goals of your organization and the needs of your business partners in focus at all times. A close partnership between the technology group and the business unit is essential for success. Data warehousing is really about making high-quality, consistent data easily accessible by business users and analysts, so they can turn this data into the meaningful information that will ultimately make organizations more successful.

(2) Set realistic goals and keep the focus relatively narrow. Designing and implementing a data warehouse is a complex process that can span months, or even years. Setting manageable goals and remaining focused on those goals during each phase of implementation is very important. By starting small, both in the size of the data warehouse and the initial number of users, you are better able to maximize the opportunity for success at each step.

(3) Establish criteria to guide you during the implementation process. Whether during the analysis phase, when you are trying to decide what data elements to include in the warehouse and what applications to support, or, during the evaluation phase, when you are trying to determine the optimal hardware and software configuration for your
warehouses, it is important to establish criteria to help organize and guide the decision-making process.

(4) Benchmark as much as possible. Especially if you are a pioneer in your organization in data warehouse implementation (and even if you are not), it is critical to base your decisions regarding the configuration and implementation of the warehouse on hard data. Once the warehouse is up and running and in production, it becomes very difficult, if not impossible, to reconfigure hardware and/or software, so the more testing you can do upfront, the better off you will be. Try to secure a machine with enough disk space as early in the process as possible, so you can benchmark with equipment that is close to that which you will have in production.

(5) Portability and Scalability. By making portability and scalability key requirements for selecting suitable technologies, you are able to start small but know that as you grow, you will be able to add more users and more data and incorporate new functionalities, like client/server capabilities, without massive reengineering efforts. The SAS System has been a key ingredient in the success of our warehouse because it gives us tools that will help us to grow far into the future.

(6) Metadata, metadata, metadata. If the data in your warehouse is not easy to access (for users of all skill levels), then your implementation is probably doomed to failure. Metadata plays a key role in ease of data access and in the architecture of the data warehouse because it helps users understand and navigate the data warehouse. It is the foundation that you need to build a data access layer. It can also be utilized to help automate many database administration functions.

(7) Associate revenue with the warehouse. Associating, as quickly as possible, a revenue stream with the warehouse, can help to ensure the future support of management when the time comes to enlarge the scope of the warehouse.

Data Warehousing is not just the fad of the moment - it is the focal point of an architecture that is designed for information delivery. This architecture, when designed and implemented well, can help organizations stay competitive today and in the future. And that will never go out of style.