Paper 117

Practical Tips and Techniques for Building an Enterprise Data Warehouse in an IBM® Environment

John Finianos, JF Information Consultancy Sarl, Beirut, Lebanon
Jugdish Mistry, Professional Solution Providers Ltd, London, UK

ABSTRACT

Much has been said about the implementation phase of an Enterprise Data Warehouse (EDW) project in general terms. Those who are directly involved in this phase appreciate the real value of knowing a few practical tips and techniques in speeding up deliverables and streamlining the whole implementation phase from source access to EDW load.

This paper will discuss essential tips and techniques that have been practically acquired during the implementation of a SAS® based data warehouse in the banking industry. Techniques such as splitting libraries, overcoming the 5 volumes limitation of SAS libraries, synchronizing EDW load processes, collecting process log data for reporting and control, building and managing format or lookup tables, techniques for space estimation, calculations and monitoring, naming conventions, filtering data, in addition to other nifty utilities built on the job. This EDW was built on an IBM mainframe running the MVS operating system using SAS versions 6.09 (MVS) and 6.12 (PC).

Key SAS products used SAS\BASE, SAS\WA, SAS\CONNECT, SAS\ACCESS to DB2, SAS\ACCESS to IMS, SAS\ACCESS to PC file formats, SAS\FSP.

INTRODUCTION

Many books have been written that outline the methodology for building a Data Warehouse, but few take it to the next step and provide practical tips and techniques to actually building one, let alone mentioning the problems that may be encountered whilst actually implementing the build.

Physical constraints are rarely mentioned, depicting the implementation of a Data Warehouse as complicated but nonetheless a straightforward process. However, during the actual implementation phase, overcoming physical constraints, building tools and techniques play a large role in the successful installation and usage of a Data Warehouse.

This paper will discuss essential tips and techniques that have been practically acquired during the implementation of a SAS based data warehouse in the banking industry. Techniques such as splitting libraries, overcoming the 5 volume limitation of SAS libraries, synchronizing DW load processes, collecting process log data for reporting and control, building and managing format or lookup tables, techniques for space estimation, calculations and monitoring, naming conventions, filtering data, in addition to other nifty utilities built on the job.

OBJECTIVE

To build an Enterprise Data Warehouse for National Bank of Kuwait (NBK) in Kuwait, on an IBM mainframe, using the MVS operating system. The final destination of the warehouse is DB2, however SAS would be used to read operational data from heterogeneous source systems, scrub, cleanses, build the warehouse. Though the ultimate repository is DB2, SAS was used as a staging area for the entire warehouse in the 1st year of development.

DESIGN

The warehouse would encompass the bank's entire business, designed and built in such a way that the warehouse could grow as the bank grows with minimal maintenance and ease. New banking products could be added, old ones removed, period histories maintained which could provide useful information such as profitability.

Flexibility was achieved through the design of the entity relational model that defined the bank's business; its products; customers etc. Currently the retail business portion of the bank has been implemented. However, the model actually encompasses the commercial business also.

The initial time spent on the model design has already paid off as the bank can introduce its new business (such as fund management) and slot it straight into the model.

USAGE

One of the prime uses of the warehouse and its data is to develop and improve the bank's Customer Relationship Management (CRM). The “7 P’s” as it is known: Promotions, Persistency, Performance, Profiling, Profitability, Prospecting, and Products.

DATA STATISTICS

A single months load would obtain data from IMS and DB2 RDBMS as well as flat files all in all 54 different sources would be used to load the warehouse. A single month load would occupy 3.3GB of DASD. The warehouse would store 36 months data online.

PROJECT TEAM

Two full time staff from the bank, a senior analyst, and a junior analyst (both originally with no knowledge of SAS) to provide:

- Business rules for transformation and mapping of data from source systems to warehouse entities.
- Validation of the data loaded into the warehouse.
- Overall control and project management of the project.
- Liaisons with other departments and systems development to provide knowledge of in house systems and existing data.

Two SAS Consultants to:

- Implement and build the warehouse in SAS.
- Design automated processes for loading/testing of the warehouse.
- Provide in house SAS training.
- Build Data Marts.
- Control the overall physical implementation of the warehouse.
- Provide SAS Support to end users
- Liaise with Computer Operations and Technical Support for automating the source extract and warehouse load processes into the production environment
- Tune overall processes
- Analyse DASD management and requirements specification

Two independent consultants are brought in for a three days per month to

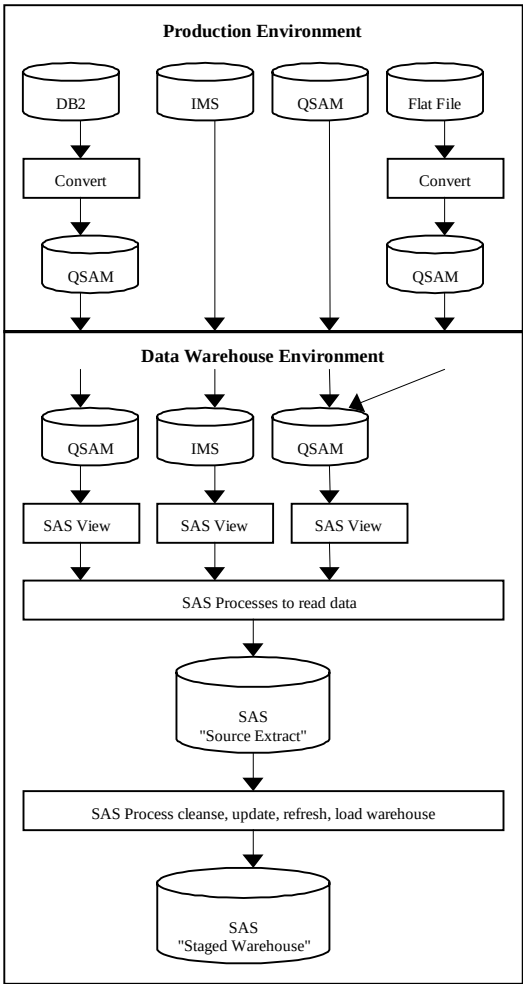
- design the logical and the physical business model
- provide high level consultancy at the project management level
- provide project level documentation

LOADING PROCESS

The diagram below outlines the flow of data from operational source systems to the warehouse. Data originates from IMS databases, QSAM files and DB2 tables. This data is read into SAS source extracts and the warehouse built from there.

Though SAS has the capability of reading the operational data directly through SAS\ACCESS and SAS\BASE views it was decided to “extract” the required data into SAS data sets before loading the warehouse. The two reasons for this are timing issues and source testing. It takes approximately 8 hours to build the warehouse once

data has been loaded into SAS data sets. It takes 5-7 hours to load the source extracts into SAS. During this time the operational databases are moved offline. Were SAS views to the data used, the overall loading of the warehouse would take less time. However, it would mean the operational sources would be tied up much longer. Also, in the initial stages keeping source data in SAS data set form meant testing and verifying the correctness of a warehouse load was much easier.



SOURCE LIBRARY SPLITTING

The initial design for the loading of SAS source extracts was to use a multi volume SAS library. The entire source extracts would be stored together; this made practical sense to keep all the data together in one place. This made management of this data very simple. However, it became apparent after initial test loading that this process would not be viable.

The basic problems encountered were:

i. Reading data from source systems was serialized. There were approximately 54 sources to be read, hence 54 batch jobs. As the entire source extracts were stored in one SAS library the Batch jobs would stack up and wait for the SAS library to become available before it could write to it.
ii. As the jobs ran serially the loading process became an over night process.
iii. Eventually, the SAS limitation of a maximum 5 volumes for a multi volume SAS library would/could cause problems.

TECHNIQUE

Split up the data each source becoming a separate SAS library.

This offered a multitude of advantages

i. All jobs to read source extracts could run in parallel, system resources prevailing. This reduced the load process down to 3-5 hours (were more initiators made available the load would take as long as the longest load).
ii. As each source was in a separate library DASD management and monitoring became easier. Automated jobs would run to report the size of each library. If the available free space in the SAS library got below a particular threshold the library could be reallocated or enlarged. Each library can now potentially become a multi volume library should the need arise.
iii. Metadata could be used to maintain the libraries and generate code to write the JCL for the extract jobs.
iv. Monitor the space usage of each library.
v. I/O could be tuned on a per data set level to optimize data threw put.
vi. No data locking issues.

The overall cost of this implementation was the warehouse loading processes would require change. No longer could a single library reference be enough to refer to all data. This meant a one off change had to be incorporated into the SAS/WA setup.

DW LIBRARY SPLITTING

In the early stages of warehouse development, the entire warehouse was kept as a single physical entity, a multi volume SAS library. The reasons are akin to those of the source library. Keep everything together; simpler for testing, loading etc. However, for reasons similar the to source library splitting, it became very apparent that the warehouse had to be split. Transactions alone would use up an entire multi volume library.

TECHNIQUE

The approach taken to resolve this problem was to split up the warehouse, but not to the granularity of a single entity per library. The split organized the data into logical groups; all transaction related warehouse entities would be stored together, all card information together etc.

This technique offered the following advantages

i. Allow warehouse to grow to hold 36 months of data as per its design
ii. Some parallelism could be implemented into warehouse load processes.
iii. The warehouse would not be completely locked; people could still use customer information whilst transactions were being updated.
iv. Easier to monitor and maintain.
v. Each library started up using one entire volume and would grow in chunks of single volumes as and when deemed necessary. Growing by a complete volume meant the library would increase by a maximum amount each time.
vi. Monitoring programs written for source libraries could be used here also. Hence early warning indicators were already in place.

Splitting the warehouse also caused some problems; existing code referenced the warehouse using a single libname. All programs that accessed or updated the warehouse would need to be changed. Additionally, as the warehouse was stored in logical groups there may come a time when they might be split a second time into single entities.

To minimize these issues, it was decided to allocate libnames at entity level, irrespective of the fact that several libnames would point to the same physical library. This meant were we to reorganize the data in another way the user code would not require changing. Only libnames with definitions would change. A further step was taken to minimize this also by providing common JCL and SAS code that did

the allocations for users. The warehouse development team would look after its maintenance and as long as the users used this they would never hit problems.

DW DASD ESTIMATION AND SPACE CALCULATION

Though DASD is relatively inexpensive these days, the procedures required to order more could be quite involved at many installations requiring reports to management for signoff and vendor specific investigations. It is therefore important to realize that storage estimation plays an important part in DW implementation. When you fall short of DASD, it cannot be bought off the shelf in a high street computer store and could potentially delay completion of the warehouse.

One of the most important factors to include in calculations is not just the size of the warehouse, but all the peripheral space required in building and maintaining the smooth running of the warehouse. Unless an installation can guarantee (say) a volume of workspace for SAS at warehouse build time the space would need to be permanently allocated. Hence this space can never be released for fear it may never be available again. The same can be said for sort space also. The use of SMS circumvents this problem considerably but does not completely remove it.

It can be said DASD is required for the following reasons

i. Store Warehouse data.
ii. Store source extract data (temporary whilst historic data is being loaded).
iii. Store source extracts in SAS form.
iv. Store Warehouse related data, programs, JCL etc.
v. Permanent workspace whilst building the warehouse.
vi. Permanent sort work space whilst building the warehouse.
vii. Permanent work space for warehouse testing and maintenance.
viii. Work space for migration of the warehouse to a new version.

Calculating warehouse growth is no easy task; it is not a case of simply adding up the usage over one month and extrapolating for n months, to get a total figure. Other factors must also be considered:

i. Not all entities will grow at the same rate or in the same way.
ii. Record counts increase over time in source systems and hence so will data in the warehouse.
iii. Data may not be removed from the warehouse but maybe from source systems. A classic example might be a customer database. An operational system might only keep a list of active customers, where as the warehouse will store information about inactive as well as active customers.
iv. Operational systems cannot always be used as a basis of record counts coming into the warehouse. As transformations occur at warehouse build time, this might introduce new records into the warehouse thus incoming records may not match outgoing record counts.
v. Rolling 'N' months of data does not imply all data gets rolled. For some entities within the warehouse data cannot be kept on a rolling N-month basis. This data is typically static information that simply cannot have the earliest month archived and a new month loaded. Examples of such data are product information and customer information. The following questions need to be answered before making any decisions:
   - When should a customer who is no longer with NBK be removed from the warehouse?
   - When should the information of the products this customer had be removed?
   - If I am a customer, but have one closed account and have one open account, does the bank still want to keep the information about the closed account? If so for how long?
vi. Net growth needs to be considered (i.e. additions – deletions + some growth due to new business) Example:
   Historically transaction counts have increased in a range of 15-20% per year. Taking the worst case (20%) by the year 2000 we will have approximately 3.5 million transactions per month as opposed to 2.4 million currently. Uplift this by 0.5 million to allow for currently unmapped transactions.

   Therefore by the end of 2000, the warehouse will have dropped 24 million records, but added 48 million records; hence net growth of 24 million records. These 24 million transactions will require an additional 4.6GB of DASD.

   The table below shows estimated transaction counts (Millions) and their DASD requirements (GB):

Year End	Dropped Trans	Added Trans	Net Growth	Additional DASD Requirements
2000	24	48	24	4.6
2001	30	57	27	5.4
2002	35	68	32	6.2

vii. Additional storage used by creation of indexes must not be ignored.

Data entering the warehouse can be characterized in the following ways:

- Replacing existing data
- Updates and appends to existing data
- Appends to existing data

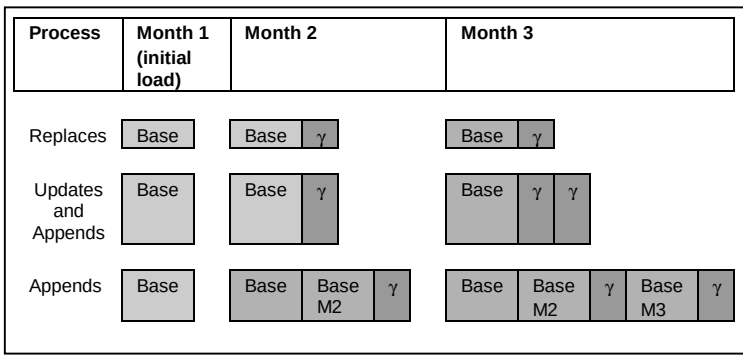
Each entity within the warehouse will adhere to one of the characteristics outlined above. Based on this, space requirements for the warehouse can be estimated. Figures used can be based on actual data loads or on business knowledge gathered. This is somewhat tricky, as it cannot be said whether data loaded thus far are representative of business volumes. However, if no other information is available these will have to suffice.

To simplify things a linear algorithm has been used to extrapolate the space required. This may not be best method but will give a good indication of space requirements as long as data growth can be considered linear.

The algorithms used are described below

Process	**Starting Size**	**Increment per month**
Replaces existing data	Base	10% of base
Updates and appends to existing data	Base	monthly increment calculated from actual loads (+ some % increment)
Appends to existing data	Base	Base (+ some % increment)

Where: Base is defined as a single month's load.
This can be viewed graphically as per below



Once all the calculations have been done, they must be uplifted by 25%-30% to allow for indexes.

The above method will give good indication of the amount of DASD that is required but it should be remembered that this is all estimation and could change dependent on factors beyond anyone's control.

The key thing is to monitor and report DASD usage at regular intervals, so potential problems are discovered well in advance of them causing any problems or embarrassments.

DW LOAD PROCESS VECTOR

An important phase in data warehousing is the process of loading data from various operational data sources into the entities of the business model after applying mapping and transformation rules. A data warehouse load might involve many of such processes running in a certain critical order to insure a successful load. If for any reason, and things do happen, a process is run out of order before its pre-requisite processes are successfully completed, there can be serious implications on the loaded data and its quality. The trick is then to implement techniques to insure that such mix up in process order will barely happen, and if it does, the process should never finish successfully. Presently, there are different techniques relying heavily on the host platform and its limitations This section describes a general technique that is platform transparent and easy to implement using the various existing SAS base tools.

TECHNIQUE

One technique to enforce the proper load process order is to rely on a dependency table and process vectors.

A dependency table is simply a SAS data set containing metadata information about each process and its pre-requisite processes. There will be one entry for each combination of a process and a pre-requisite process. This table is maintained by the Data Warehouse administrator. The structure of this table is:

```
process length=$8 label='DW Load Process'
prereq  length=$8 label='Pre-Requisite'
```

A process vector is a single entry in the process vector table. The process vector table is a SAS data set which is emptied or reset at the beginning of a periodic DW load and set at the end of each process. The structure of this table is:

```
process length=$8 label='DW Load Process'
datetime length=8 label='Completion Time'
```

This technique is implemented in four consecutive steps:

i. The process vector table is reset of all its entries at the beginning of a periodic load.
ii. The first step of each process is to determine its pre-requisite processes from the process dependency table.
iii. The second step of each process is to check the process vector table and make sure that the process vectors of its pre-requisite processes have being set. If any process vector of the pre-requisite processes is not set, then the process aborts with an abend error.
iv. The last step of each process is to set its process vector in the process vector table after a successful completion. Any process failure will prevent the setting of the process vector.

Step i, iii, and iv is easily implemented using the SAS Macro tools . Step ii is implemented using the SAS/SQL procedure.

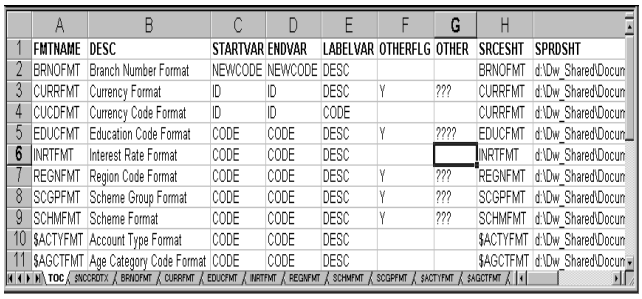
FORMAT TABLES MANAGEMENT

SAS format tables are powerful tools used heavily as efficient means to standardize and transform source data. The number of these tables can grow depending on the complexity of the data warehouse project and its transformation rules. As the number of the SAS format tables grows, the challenge is then to find an easy way to maintain and manage them. This section describes a couple of techniques used to manage and maintain a continuously growing list of SAS format tables.

TECHNIQUE 1

SAS formats can be grouped into two categories: descriptive - used to interpret codes, and transformational - used to standardize and transform codes. The data sources for each of the two groups are maintained in a MS/Excel spreadsheet with a worksheet for each source. However, the first worksheet in each of the spreadsheets is a table of content with enough information or metadata about each source to build the corresponding SAS Format.

The format metadata (see next figure) includes the format name, description, start and end range variable names, label variable name, other value information, and any pertinent information needed by the PROC FORMAT procedure to build the format.



	A	B	C	D	E	F	G	H	
1	FMTNAME	DESC	STARTVAR	ENDVAR	LABELVAR	OTHERFLG	OTHER	SRCESHT	SPRDSHT
2	BRNOFMT	Branch Number Format	NEWCODE	NEWCODE	DESC			BRNOFMT	d:\Dw_Shared\Docum
3	CURRFMT	Currency Format	ID	ID	DESC	Y	???	CURRFMT	d:\Dw_Shared\Docum
4	CUCDFMT	Currency Code Format	ID	ID	CODE			CURRFMT	d:\Dw_Shared\Docum
5	EDUCFMT	Education Code Format	CODE	CODE	DESC	Y	????	EDUCFMT	d:\Dw_Shared\Docum
6	INRTFMT	Interest Rate Format	CODE	CODE	DESC			INRTFMT	d:\Dw_Shared\Docum
7	REGNFMT	Region Code Format	CODE	CODE	DESC	Y	???	REGNFMT	d:\Dw_Shared\Docum
8	SCGPFMT	Scheme Group Format	CODE	CODE	DESC	Y	???	SCGPFMT	d:\Dw_Shared\Docum
9	SCHMFMT	Scheme Format	CODE	CODE	DESC	Y	???	SCHMFMT	d:\Dw_Shared\Docum
10	$ACTYFMT	Account Type Format	CODE	CODE	DESC			$ACTYFMT	d:\Dw_Shared\Docum
11	$AGCTFMT	Age Category Code Format	CODE	CODE	DESC			$AGCTFMT	d:\Dw_Shared\Docum

TOC | $NCCRDTX | BRNOFMT | CURRFMT | EDUCFMT | INRTFMT | REGNFMT | SCHMFMT | SCGPFMT | $ACTYFMT | $AGCTFMT

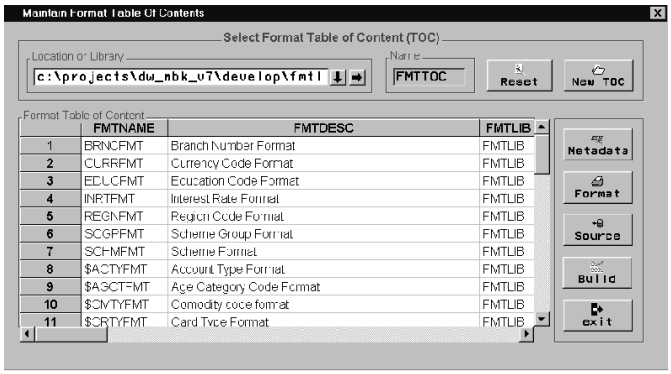
This technique is used mainly to manage and organize the various formats.

A set of specifically written macros are used to build the SAS formats from their Excel sources, XCL2DSET() to import Excel data to a SAS data set, DSET2FMT() to build a SAS format from a SAS data set, and XCL2FMT() to build a SAS format from Excel data by using the previous two macros. So the process of building or rebuilding a format is as easy as calling the macro XCL2FMT with the proper parameters as outlined in the format metadata table.

```
%macro xcl2dset(sprdsht,worksht,dsname);
%macro dset2fmt(dsname,fmtlib,fmtname
              ,startvar,endvar,labelvar
              ,otherflg,other);
%macro xcl2fmt(
       sprdsht   /* spreadsheet */
      ,worksht  /* worksheet */
      ,fmtlib   /* format catalog */
      ,fmtname  /* format name */
      ,startvar /* range starting var */
      ,endvar   /* range ending var */
      ,labelvar /* label variable */
      ,otherflg /* other value flag */
      ,other ); /* other value */
```

TECHNIQUE 2

The second technique in managing formats is to write a front end SAS/AF application that uses the format metadata to manage and build formats. The next figure is a snapshot of such an application main screen. This application uses the macros described in the previous technique, except it supplies the macro parameters automatically from the format metadata. This application allows the user to edit the metadata table, browse an existing SAS format, edit a format source data, and build or rebuild one or more SAS formats.



DW PROCESS LOG DATA

Gathering information about the DW loading process is a core step in providing the warehouse administrator with the ability to tune up, quality control, and report on the nature of any of the loading processes. Such information might include the process start and end time stamps, number of records modified and added, record length, number of variables, and other information pertaining to the structure of the loaded entity. The ADDLOG macro is written to gather the loading process information:

```
%macro addlog(
      process  /* Loading process name */
     ,entity   /* Loaded entity dataset */
     ,logloc   /* Log data set location */
     ,logname  /* Log data set name */
     ,logmsg ); /* Log message  */
```

The ADDLOG macro is called at the start and the completion of a loading process in order to save the process time stamps and the DW loading entity content information from the SAS dictionary library. The log data set can have the following structure:

```
LIBNAME  length=$8  label='Entity Library'
MEMNAME  length=$8  label='Entity Name'
CRTDATE  length=8   label='Creation Date'
MODATE   length=8  label='Entity Mod Date'
COMPRESS length=$1  label='Compressed?'
NOBS     length=8   label='Number of Obs'
NVAR     length=8   label='Number of Vars'
OBSLEN   length=8   label='Obs Length'
LOGMSG   length=$30 label='Log Message'
DATETIME length=8   label='DateTime Stamp'
JOBNAME  length=8   label='Job ID or Name'
LOADPERD length=4   label='Load Period'
```

From the above data set, various reports can be generated regarding the nature and the scope of the loading process. One important report is generated after a full load is completed to list all the successful loading processes with the total process executable time, number of observation added in comparison to the previous load, and a flag to alarm any addition of new variables and unexpected change in observation length.

DATA TESTING AND SCRUBBING

It is essential to adopt an early testing strategy and put it into place before populating the EDW. This strategy should be dynamic and open for continuous monitoring and improvement.

Before writing or designing a test, it must pass the '*able*' test i.e. a test must be achiev*able*, repeat*able*, manage*able* and must be meaningful.

There are many different types of tests on data, all of which have different functions. Some tests are listed below.

Type of test	Usefulness
Existence	Make sure what we want is there
Validation	Data has got correct values
Rough Test	Range checking of data (sensibility tests)
Totals/Counts	Business test on record counts that validate the correctness of loading.

Note: all the above tests are tests after the event, i.e. once the warehouse has been loaded.

It is also essential to test source data, before it enters the warehouse. This will trap and highlight potential inconsistencies within the data before they impact the warehouse. This can be achieved via the use of a temporary staging area where data to be transformed into the warehouse are placed.

Tests such as existence of key fields and correctness of data can then be applied.

It was found useful to adhere to the below rules when writing tests

i. Each test must be autonomous, i.e. it must work in its own right without need of any other code.
ii. Each test must return a result, pass, fail with a reason code. Ideally this result should go into a data set.
iii. Perform hierarchical tests only when necessary.
iv. It is useful to use binary numbers for return codes if performing multiple tests in a test suite. Each test can then set a bit in the number indicating which test failed.

OTHER NIFTY TIPS

A list of the important tips that saved time and made life easier are listed below:

- In order to save time and effort determining which process is loading which entity, it was found beneficial to give the loading process the same name as the loaded entity.
- To reduce data entry error during the specification of the loading period macro variable, it is advisable to localize the setting in one file to be referenced by the various data extracting and loading processes. Also, a macro can be written to set up a variation on the format of the loading period to meet various data filtering requirements, such as YYYYMM, YYMM, DDMMMYYYY, etc.
- Usage of the SAS option ERRORABEND stops cascading of errors and proliferation of empty data sets.
- Usage of metadata, as much as possible, increases the flexibility and the speed in which changes can be made at many stages of warehouse implementation.
- Compiled macros save time.
- Documenting known source and warehouse data problems helps the end user in understanding the nature of the data.
- Implementation of change control procedures at the early stages of EDW implementation is imperative.
- It is essential to provide education on the usage of the warehouse data before the EDW is released.
- Hierarchical jobs can be serialized if given same batch job name.

CONCLUSION

To increase the success rate of a DW project, a good business model, careful planning and, the right level of implementation skills and tools are required at the various stages. This paper intended to add some useful tips and techniques to the DW project planning and toolbox, such as data storage considerations and estimation at both the source and DW level, lookup and transformation table management, loading process control, data testing, filtering, and others. Given the right skill mix, the success rate of an EDW project can be increased dramatically on a timely basis.

REFERENCES

ACKNOWLEDGMENTS

The authors wish to thank the entire National Bank of Kuwait EDW team for their professionalism, dedication to the project, and their assistance in the process of writing this paper.

CONTACT INFORMATION

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

John Finianos
JF Information Consultancy Sarl
Beirut, Lebanon
Tel: +961 (0) 3 713430
+965 (0) 2 422011 x3122
Email: JFIC@cyberia.net.lb

Jugdish Mistry
Professional Solution Providers Ltd
62 Stanhope Rd
Cippenham, Slough
Berkshire SL1 6JS, UK
Tel: +44 (0) 1628 603 428
Fax: +44 (0) 1628 541 561
Email: jugdish@pspltd.demon.co.uk