Paper 114-29

Next Generation Data Warehousing with SAS 9

Gary Mehler, SAS, Cary, NC

ABSTRACT

This paper covers advanced data warehousing topics that can be performed with SAS 9 and explains how these approaches can be used as well as how they work These topics will show how a basic data warehouse can be turned into a full-scale production system with integrated data integrity handling, success/failure processing, and utilizing slowly-changing dimensions. These capabilities allow production-quality warehouses to be constructed within the graphical user interface of the SAS ETL Studio design environment.

INTRODUCTION

Data warehousing is growing more commonplace as more organizations see the benefit of delivering rich business intelligence to their business decision makers. As such, more people need to know more about the basic approach as well as advanced topics. This discussion follows on a paper discussing introductory topics in next generation warehousing (*Next Generation Warehousing with Version 9, 2003)* which covered the foundational elements of a modern data warehousing platform. These topics will be briefly reviewed here:

- ***Metadata*** – A common, open, multi-user shared repository used as the metadata storage for all warehousing and business intelligence processes. Metadata is available for re-use by other applications, and has a built-in change management system facilitating a check-in/check-out system with history and audit trails.

- ***Multi-level planning*** - Collections of repositories can be combined to store metadata as needed allowing the reuse of foundational metadata. These collections can be used in a promotion-based management scheme of Development-Testing-Production levels as well as be replicated as needed for performance needs.

- ***Multi-user administration*** – An integrated management console brings together many administrative functions related to metadata as well as application and data services. Security and user access privileges can be set and maintained.

- ***The ETL design process*** – Metadata is defined that describes the data Extraction, Transformation, and Loading processes that move and reformat data values to meet the business needs for which the warehouse is being developed. Wizards help the source and target definition process, and an easy-to-use templated process designer is used to design data flows. Advanced transformations are available to join, split, filter, and cleanse the data as needed.

- ***Extensibility*** – In addition to using built-in transformations, the warehouse designer can add their own transformations into the mix, either in SAS code-based form, or as Java plug-ins to ETL Studio. In either form, these are added to the library of available transformations that can be used in a standard way in an organization.

- ***Data Quality*** – The ability to standardize data values and use fuzzy logic to determine match codes is a key data quality function. Also, other data quality functions are available for use in general ETL processes.

- ***Common Warehouse Metamodel (CWM)*** – Adherence to a standard for relational metadata means that metadata from data model design tools or other warehousing tools can be imported and exported as needed. Bridges allow sharing of metadata with other applications such as modeling tools or other warehousing applications.

- ***Other sources*** – A library of operational system data surveyors is available to handle data access from systems like SAP, Siebel, PeopleSoft, and Oracle Applications. This is in addition to the very wide range of data sources supported by the SAS system, including databases, desktop application files, text formats, as well as ODBC and OLE-DB based data.

- ***Deployment*** – Integration with a full-function scheduling package means it is straightforward to schedule, maintain, and monitor processes that are run on a regular basis. This adds more load-balancing and parallel execution capabilities to the deployment realm.

This set of functions can be considered a minimum for modern warehousing functionality. It provides the ability to define, manage, and share data using consistent transformations for basic warehouse needs. It also allows larger groups to cooperatively contribute to a larger-scale warehousing project.

REAL-WORLD WAREHOUSING CHALLENGES

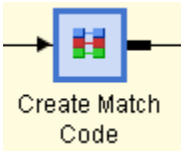
In addition to the enabling topics listed above, the real-world imposes another set of challenges to get work done in a reliable fashion. Issues like the integrity of data values due to processing errors or other events need to be handled before they snowball into generally unreliable data. Speaking of processing errors, how are individual error or exception cases tracked and handled? Dealing with advanced data management topics like Slowly Changing Dimensions (SCD) imposes an additional challenge for the warehouse developer, as these can be complicated algorithms to code accurately and efficiently. Without specific design tool support for complex transformations like SCD, handling changes in dimensions is very difficult for a basic warehouse developer to perform.

ETL Studio, the SAS 9 data warehousing design environment, includes features to handle these advanced needs. The remainder of this paper will show how these capabilities can be used to more easily solve the needs in these areas as you develop a data warehouse. Along the way, business practice recommendations are made to help in understanding how these possibly complex issues can be handled effectively.

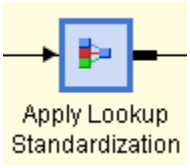
HANDLING INCONSISTENT DATA

Various studies have shown that poor quality data lead to increased cost and decreased reliability of business intelligence results. Poor quality data can be of various types, including data entry errors or simple issues like having missing or out-of-range values. Different approaches can be taken to resolve these quality issues ranging from advanced use of fuzzy-logic matching to simple missing value checks. As incoming data quality issues should be considered and treated first, we'll cover data quality first, then discuss data integrity validation considerations.

DATA QUALITY



The advanced use of fuzzy logic is provided through SAS Data Quality products to allow syntactically different data values like "John Smith", "Mr John Smith", and "John J. Smith" to be seen as matching the same value based on smart name-matching logic and the use of other fields. In this case, a match code is generated and used to help determine which different data values can be considered to match. The first step to this is to generate a match code using a data quality-based matching function. The transformation in SAS ETL Studio for match code creation has access to a central Quality Knowledge Base, where a data quality steward can define and manage the basic quality settings that can be used by ETL developers.



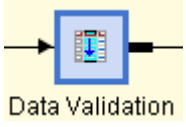
The second step is to standardize values to bring the resulting data to a standard, or preferred data value. In the above example, if we choose to err on the side of completeness, the resulting standard name field might end up as "Mr. John J. Smith". All related "matches" would show up in output data with this name. The Quality Knowledge Base is again used here since we'd like to bring a large library of standardization metrics to bear to solve this problem. Smart matches are available for names, addresses, organizational units, and other common data types.

Following from this example, if we needed to perform a calculation for each separate customer in our warehouse, we would want to be sure to perform the standardization before performing important analyses.

This approach would also facilitate the process of de-duplication in a warehouse. De-duplication is the process of removing records that are largely duplicates of other records. So, if Mr. Smith's example can be used again, suppose a data entry person records a new transaction from a "Mr. J. J. Smith" and doesn't find an exact name match in our database so proceeds to define a new customer record for him. A later data review might find a "Mr. J. J. Smith" at the same address as "Mr. John Smith", so the two records can be collapsed into one master record.

In the cases described here, these advanced data quality operations can be performed on data as it enters the warehouse, and "good" quality data can proceed further through the warehouse to good effect. Other standard types of cleansing operations can be performed on other well-understood field types such as address or organization names.

BASIC DATA INTEGRITY VALIDATION



In addition to the advanced fuzzy-logic algorithms, basic data integrity validation can be used as general-purpose transformation steps in many, if not all, transformation flows in a warehouse. This choice can be based on expected data volume through any particular flow, and offset with the anticipated cost of dealing with invalid data values downstream in other warehouse processes. However, when dealing with dimensional data models, the relationships between primary and foreign keys are imperative to maintain. Integrity validation can aid in this function if they are applied consistently.

Filling in missing values automatically

As an example, missing values can be acceptable in some statistical calculations, but if a replacement value can be determined from other values data fields, some cleanup could be warranted. Let's assume we have a table of incoming transactions and need to process it for general reporting needs. Figure 1 below shows a typical flow, which contains two stages: processing raw data into a staging area, and subsequent post-processing to generate a dimensional table. In this example, both the staging and dimensional table are very similar. The main difference is that we will validate data integrity and apply a slowly changing dimension process to the data after data leaves the staging area.

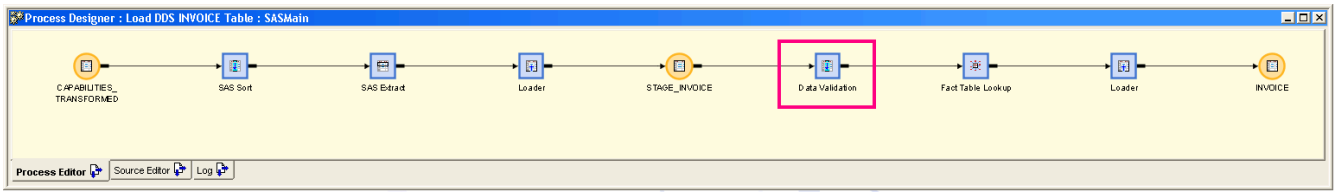


Figure 1. A basic data flow in ETL Studio's Process Designer. In this flow, data is transformed through a series of steps from source table to its target. As part of this flow, data is moved from source to a stage area, after which a Data Validation transformation is included (shown in the box).

Depending on how the data is entered or makes its way to us, we could end up with missing values for an important variable: source system code (where the transaction originated). In this case, since the data is first encountered in the ETL process, inserting an 'ETL' tag can provide some useful information later in processing. In Figure 2, we see a panel in which some settings are displayed dealing with missing values. We see that the SOURCE_SYSTEM_CD column will be inspected to check for a missing value. If found, it will be changed to the string "ETL" through this validation. This will allow subsequent steps to perform sorting, grouping, or other operations as needed for useful analysis.

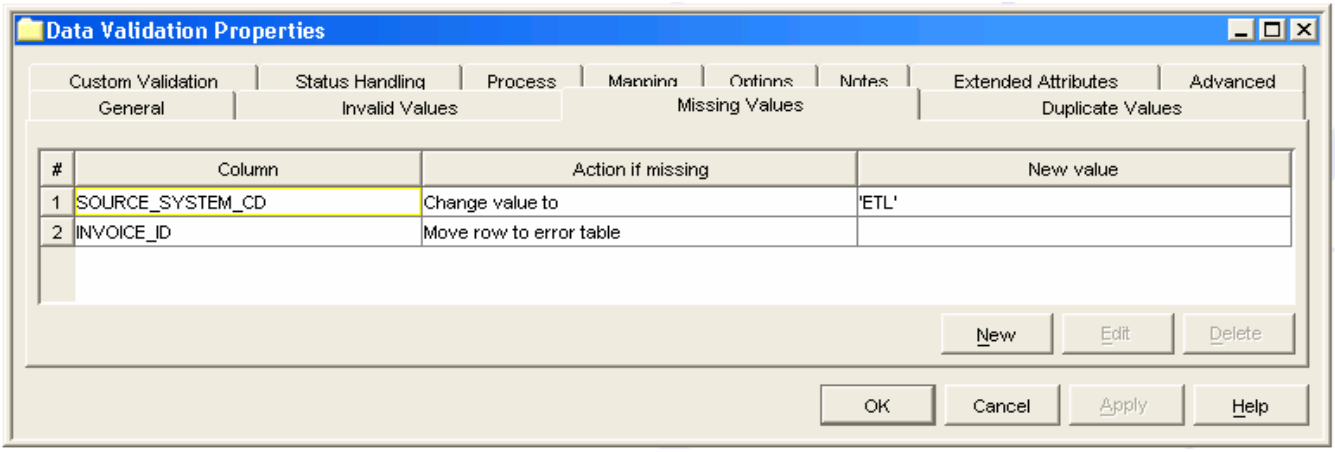


Figure 2. Inspecting for and Handling Missing Values in a Data Validation transform.

Filtering incomplete records for later processing

As another example, assume that missing values are present in transactions for the Invoice ID. In this case, it would be difficult to process this information at all, so it may be wise to set these types of records aside for later hand processing. In Figure 2 above, we see that the validation transform has been configured to check the column INVOICE_ID, and missing values will be placed into a special table for later processing. These "kick-out" records would need to be accumulated in a work area for manual processing and have subsequent records appended to the working table of items needing further attention. This is what is done in this transformation to ensure that no data is lost. As a follow-up, the warehousing staff would review the kicked-out records periodically to determine their disposition. However, the data that passed through this transformation phase successfully can be assumed to be of high quality and ready for downstream use.

Aborting a process when something serious shows up in the data

For the final example in this section, let us assume that certain types of data integrity problems are indicative of a major problem and that their presence suggests a general processing failure. In our transaction example, this might be triggered when a transactional record for an order has an invalid invoice type code. When this occurs, it might suggest that some major failure has occurred, for example, an inability to read the current version of the invoices database table. Figure 3 below shows how this could be handled. In this case, seeing any unsupported INVOICE_TYPE_CD would cause the entire job to abort at this point could help to prevent the target table from being populated with out-of-date data. So, the ability to specify a processing flow change based on critical data flaws can help prevent problems from becoming more serious reporting problems.

The figure also shows that two other columns are processed to look for values that are incorrect, but themselves only indicative of problems with those particular data records. In those cases, just moving the data out of the way into a "kick-out" area prevents the bad records from proceeding, but allows other data to flow through unimpeded.

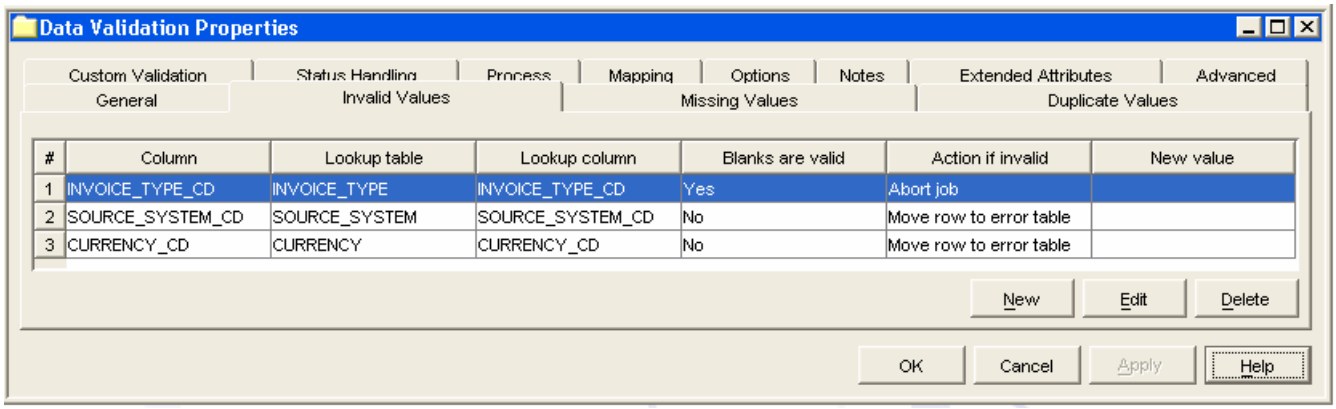


Figure 3. Data Validation for Invalid Values. If the column contents aren't of the allowed types, processing can abort, or have data moved out of the main flow.

The concept of a "kick-out" area is a powerful one, and can be set up as a special library in the warehouse. Individual tables can contain a running set of records needing special handling before they're allowed back into the warehouse. This can be thought of as a "time-out" area for bad data and prevents known invalid data from skewing analysis processes like constructing OLAP cubes. In the example above, the error table into which invalid rows of data are moved is specified as an option (under the Options tab in the Figure 3).

The data validation transform has capabilities of using lookup tables to verify or replace values. In the case in this example, we are validating three source table columns (INVOICE_TYPE_CD, SOURCE_SYSTEM_CD, and CURRENCY_CD) in three separate reference columns in three different lookup tables. These columns were chosen because they have foreign key relationships to a star schema fact table, and we want to ensure the referential integrity of the keys before using them. These reference tables can be used to specify ranges of allowed values that are known to be good keys. Just as a kick-out library can be set up in a warehouse as described in the previous section, it's a good idea to set up a reference library in which all master lookup tables are stored. One last point: there's no real point in performing both an invalid value check and a missing value check on the same data columns; missing value checks are helpful on primary keys, while foreign keys benefit from validity checks, so that's a good rule of thumb to keep in mind when processing.

Custom validation

The full power of the SAS language is available to help describe any type of condition that can occur in the data, as well as the result handling after the condition is triggered. This is useful when it is simpler to specify an exact condition rather than looking up values in a table. Figure 4 below shows how this can be specified. When placeholder values are encountered, special processing code can be engaged to modify or otherwise handle those data records in a special way. This is a straightforward way to allow placeholder values to propagate through some types of processing where they might be "don't care" values, and allow processing based on them at a later point when they can be resolved completely.

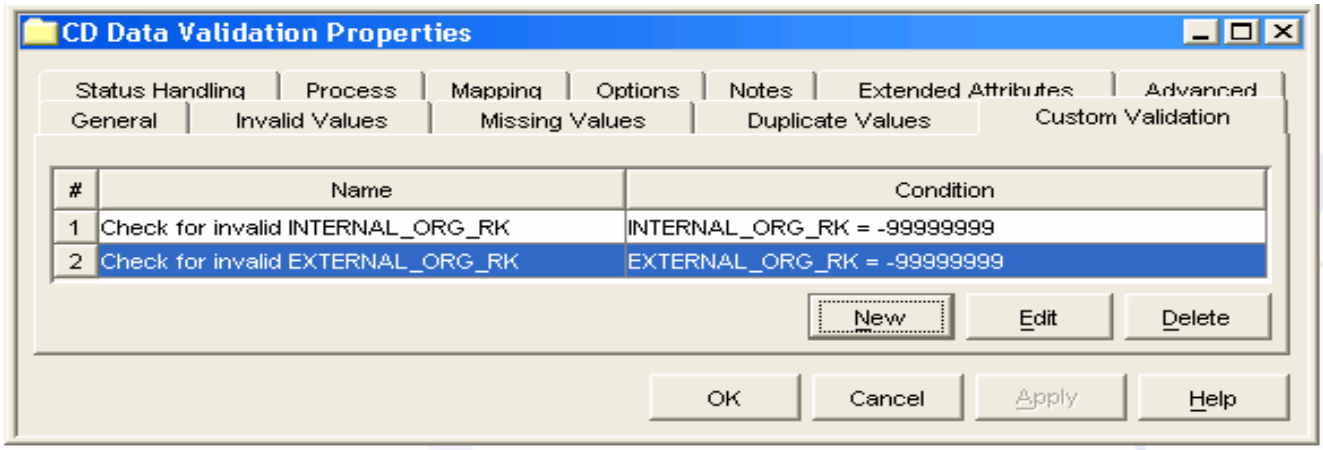
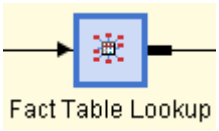


Figure 4. Custom data validation showing that placeholder values (-99999999 in this case) trigger special handling.

Lookup tables



Another way to handle data values is with table lookup transformations. In this type of transformation, a separate table is used to specify the source and target values of a data value in a record. This is commonly used when working with star schema data. In the star schema topology of a central fact table and associated dimensional tables, it is a common activity to look up an item from a fact table and do a translation based on it.

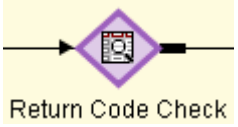
Although not shown in operation here, it is another graphically-driven transformation in which a lookup table is specified, as well as the lookup and replacement columns. In a lookup, a column value can be replaced, or the column value can be used to determine the value of a different column. A common lookup activity is to replace numeric codes with text values. So, if we are storing country codes in a numeric column, a simple transformation can fill in the values of a text column if a human-readable form is also needed. In a star schema, this type of lookup is used to find the primary keys referenced by foreign keys.

DATA INTEGRITY SUMMARY

This section has discussed how two types of data analysis approaches (data quality and data integrity) can be used to keep data flaws from causing major problems in a data warehouse. Their use needs to be planned in to ETL flow design. The safest approach is to determine whether each and every ETL flow needs to treat data flaws individually to prevent data problem propagation. This also needs to be weighed against expected data volume since these checks aren't free. In high-volume scenarios, these checks can be consolidated in key points so as to not impede data throughput. When thinking about data integrity, its also a good time to consider defining libraries for error records (kick-outs) and reference tables (standard lookups).

It's important to remember that when working with dimensional models or star schemas, there are a lot of inter-table relationships that have to be exactly correct for processing to be able to proceed as desired. When data arrives from multiple sources or is updated with varying schedules, inconsistencies can arise. The foreign key – primary key relationships are vital to maintain with integrity, and the data validation approaches we've discussed here can be very helpful in that respect.

ERROR AND FLOW PROCESSING



In a perfect world, nothing ever goes wrong and there is no need to handle problem cases. All ETL flows always run perfectly in the time allotted, and there are never hardware or network outages that impede their progress. That is the world in which much planning occurs in the initial stages of a warehouse project, but that view is hopelessly optimistic in just about all cases over time.

In the real world, things do go wrong in continually varying ways and treating at least some of them in the normal course of ETL flow design is a good investment of time. There are two modes of handling this: the most obvious is to handle an error case when it occurs, but an equally important one is to gather statistics during normal runs as well. The statistic gathering case is helpful to determine a baseline for typical runs, such as time completed, or the frequency with which a table is updated.

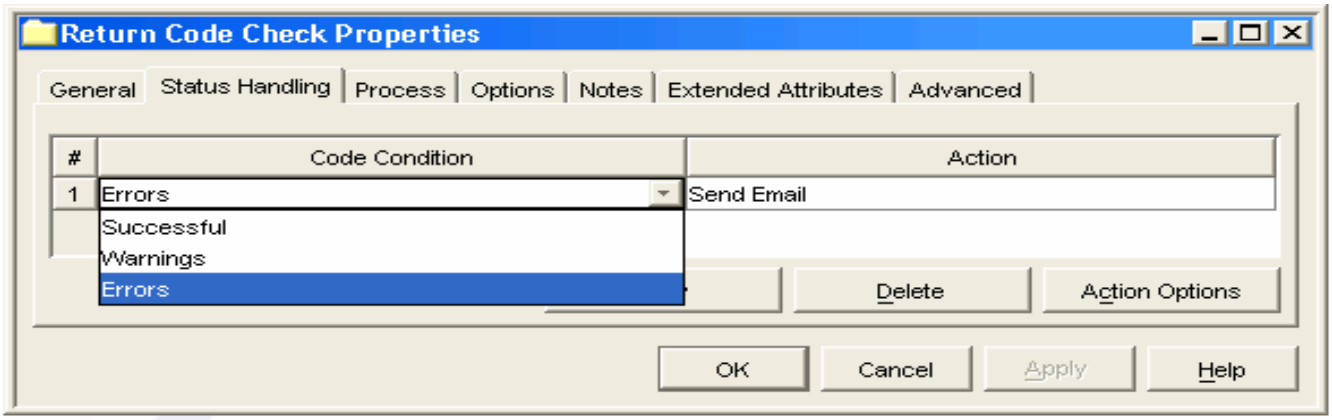


Figure 5. Specifying how errors will be handled in an ETL flow.

In Figure 5 above, we see a panel in which return code settings can be specified. It is important to think about how regular or abnormal conditions need to be handled in flows as they are being designed. The obvious cases to handle are the completely successful and the completely failed transformations. In these cases it can be easy to determine whether successful runs should logged every time for baseline setting, as well as how to handle complete failures. In complete failures, a job abort or e-mail trigger may make the most sense.

It is also important to consider how minor problems (such as warnings) need to be handled. These are often overlooked since the overall process may appear to work just fine (i.e., reasonable-looking data is generated as a result). However, warnings

can be the bane of an ETL administrator's existence when some sort of diagnostic needs to be run, as when a more serious problem crops up. In these cases, how can the administrator know what are thought to be "acceptable" minor errors versus some new condition that has a more dire consequence? The theoretical answer is to not allow ETL flows that aren't perfect to go into production use. However, that can be unrealistic in some shops, so a partial answer lies in baselining minor issues to an accumulating logfile. So, if a flow-based review is needed at some time, new issues can be distinguished from the long-running (and therefore acceptable) issues. This can help to resolve problems more efficiently. Figure 6 below shows different types of handling that can be used to monitor typical issues.

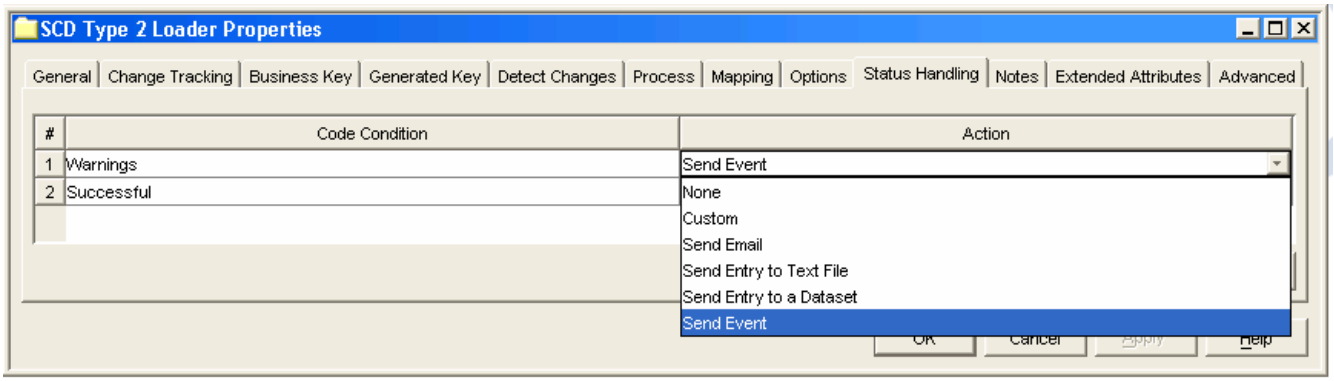


Figure 6. Specifying status handling on a table load action.

To discuss further how status values and exception reports are collected, the ETL administrator should consider the arsenal of methods at their disposal. The most serious errors could generate an e-mail message that is delivered to someone's pager for immediate processing. Less serious items can be logged for later inspection. If the inspection will occur manually, a text log that grows as more status is posted makes some sense. If the data are going to be programmatically processed, appending them to a table is a good idea. Figure 6 shows how these could be configured for different types of conditions.

So, assuming that errors will occur, which what granularity do they need to be captured? That will depend on the criticality of the function being performed. In the most critical cases, action may be needed immediately, such as in the very transformation in which a problem is detected. For less serious ones, checking and logging at critical junctures in a flow may make sense. This can be done by inserting an explicit status check transformation at that point. For minor issues or just general logging, this might make sense at the ETL flow boundary itself.

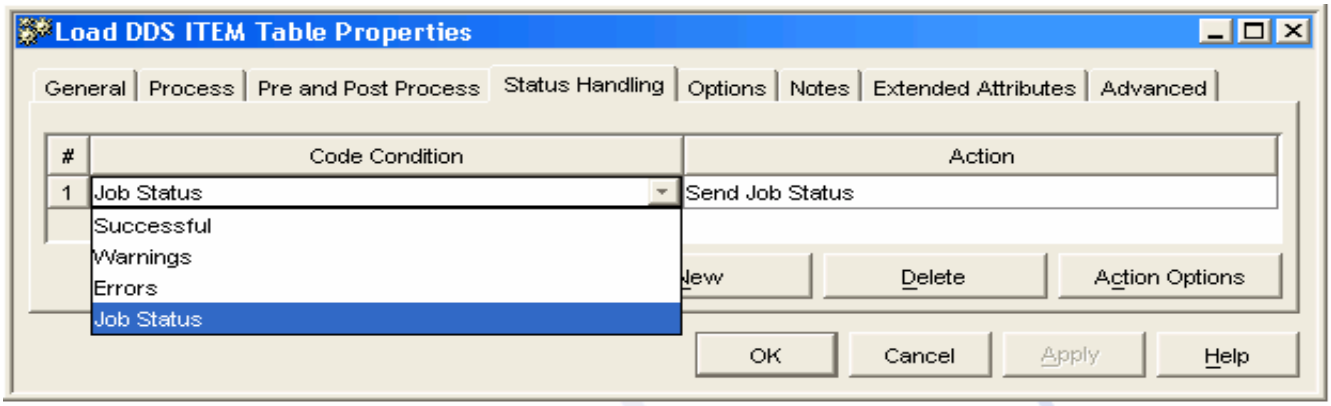
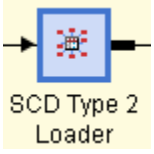


Figure 7. Status handling settings at the flow-level.

In summary, error and status handling is a very important aspect of a well-running production warehouse. In some cases, designing status handling into each ETL flow may make sense if a full record of past run status is required.

HANDLING CHANGE IN WAREHOUSE DATA



It's a known fact that data does change regularly, but there are some types of changes that are more interesting than others. This is especially true in dimensional models such as star schema (a common warehousing structure in which a central fact table of transactions or values is associated with a collection of dimension table that contain related information about business-oriented areas such as Customer, Employee, Product, or Supplier). In a star schema, a fact table will contain transactions that will generally never change. Rather, the fact table will continue to grow by having new records added. These changes are easily managed

by just appending to the fact table.

Changes to dimensional tables are more interesting to data warehousers since these records tend to stay around and need to capture whether and how these records change over time. This notion is called a "slowly changing dimension", and has been discussed at length in the literature, such as by Ralph Kimball. Capturing information about the past needs a convention so we can understand the cases in which a value is current and correct.

For example purposes, let's work with a table of items in our inventory. Any given item will have only one current product code, supplier, unit price, and other values, but may have other information available depending on how frequently these variables change. How or whether prior product code, price, and supplier information is retained defines the slowly changing dimension approach used.

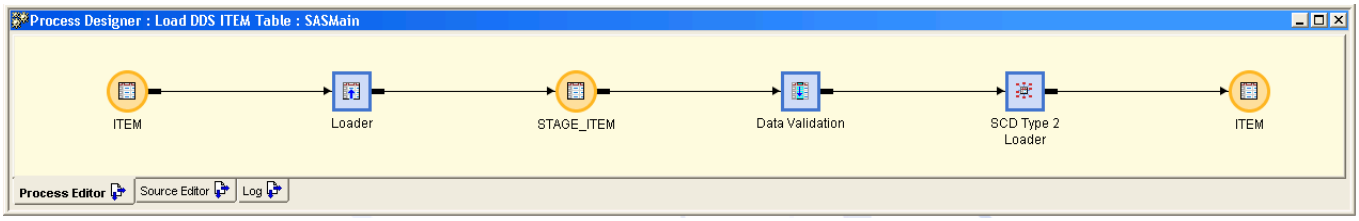


Figure 8. A typical ETL flow showing how change is managed between a staging tier and the final data store.

There are three standard and documented ways to handle slowly changing dimensions in the warehouse literature: Type I, Type II, and Type III. We will discuss each of these in turn.

TYPE 1 SLOWLY CHANGING DIMENSION
As generally defined, handling slowly changing dimensions in the Type 1 way means not caring about historical values. In this case, a table that contains the inventory item record would contain an values that only reflect the current setting and no prior information is retained. This is a fine approach for OLTP (on-line transaction processing) systems where only the current world is considered.

TYPE 2 SLOWLY CHANGING DIMENSION
This approach supports saving any number of past values of the address, and the appropriate value is selectable, typically based on an effective date range. So, we would add two columns to track the start and end date/time during which the address was valid, and the most current one would be the present address. Finding the right value becomes a more complicated SQL query, but is a straightforward activity. This is the standard approach in many general warehouses and adds extra rows to the dimensions to track the changes to the address over time.

TYPE 3 SLOWLY CHANGING DIMENSION
This approach adds an extra column to the main record and allows storage of the current and prior address.

Which approach is for you?
The Type 1 approach is simple to handle in a warehouse in that entire records (rows) of data can be overwritten. The downside is that there is no ability to do historical reporting. However, it is the fastest and easiest to implement. Whenever a table is just overwritten without any special handling, Type 1 SCD is probably being performed.

Type 3 is also straightforward in that it can be accomplished by adding an extra column to the data to retain the current (vs previous) value. In this case, new values cause a movement to the extra variable of the older value, so we'd be to retain the present and immediately prior product code.

Type 2 is the most capable but also the most complex. Kimball discusses its powerful advantages in various references. It has the capability to do show data for any point in history, and is therefore the most common in large-scale warehouses. It stores extra column information for each data record to support ranging. In Type 2, extra rows of data are added to tables to keep track of each different value of the slowly changing dimension.

In a walk through example here, we'll see how to set up and use this type of transformation allows a number of different analyses to be performed on a complete set of data for facts (transactions) over time and how they can be used with dimensions (items like the records that describe a customer over time).

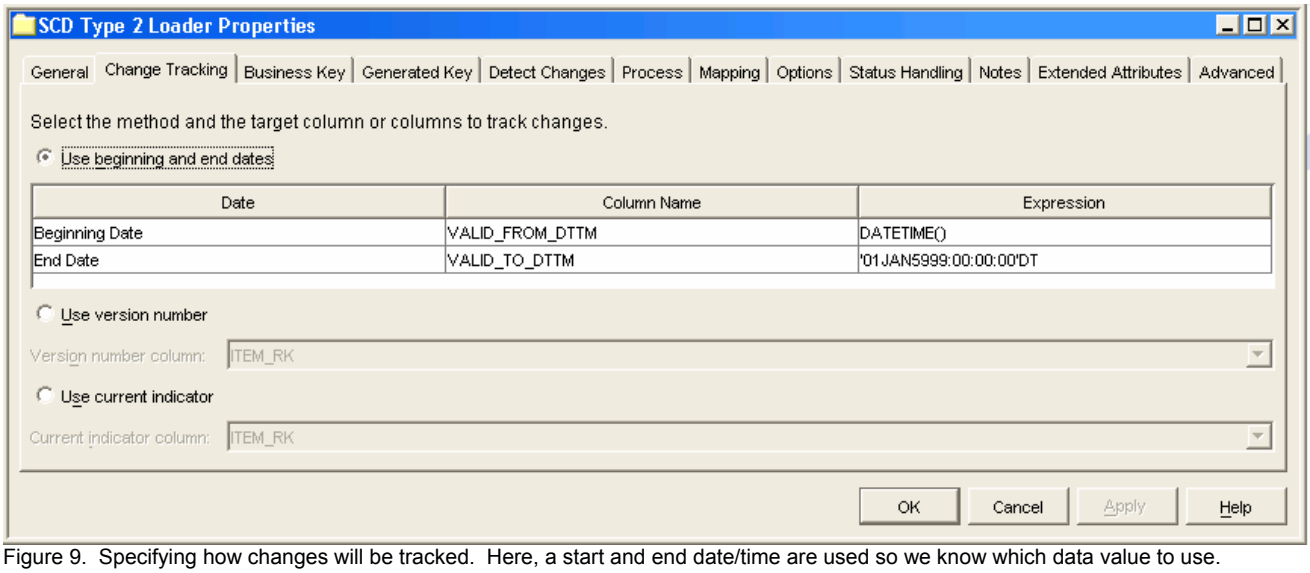


Figure 9. Specifying how changes will be tracked. Here, a start and end date/time are used so we know which data value to use.

In Figure 9 above, a date range is set up for our data. In this case, the beginning date is now and the end date is some point safely in the future. As more data points are added, the end date can be changed to the new termination date, and a new record for the current value entered. The convention used in this warehouse is to use column names VALID_FROM_DTTM and VALID_TO_DTTM consistently to ensure we know which tables are being managed with change tracking. In many warehouses, most, if not all, important dimensions will be under this type of change control, since many values can change over time.

Once a time/date range has been specified, we know how to find the desired value. Instead of just looking for a specific value in a column of data, we now will have to perform a query that indicates not only the column value, but the date in question. The results of the query should be the right values for that item. So, more specifically, we could imagine a query requesting the unit price of a particular inventory item on January 1, 2004. Since the table has date information stored (in beginning and end dates), we can determine the value for this or any other date for which we have data.

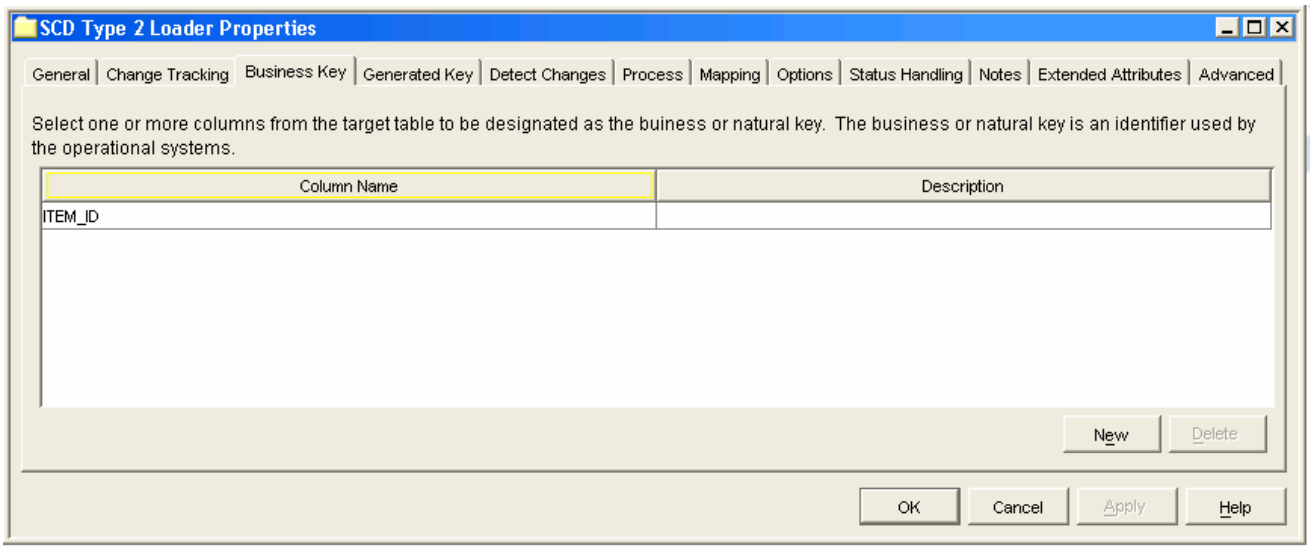


Figure 10. The Business Key is what we use to determine what we are looking up.

The Business Key can be all of the non-internally-used primary keys of our tables. In this example, ITEM_ID is our business key. The business key is different from the primary key, since we're using a surrogate (numeric) key in our example. This allows us to merge data from varying source systems without having to worry about primary key collision and growth issues over time.

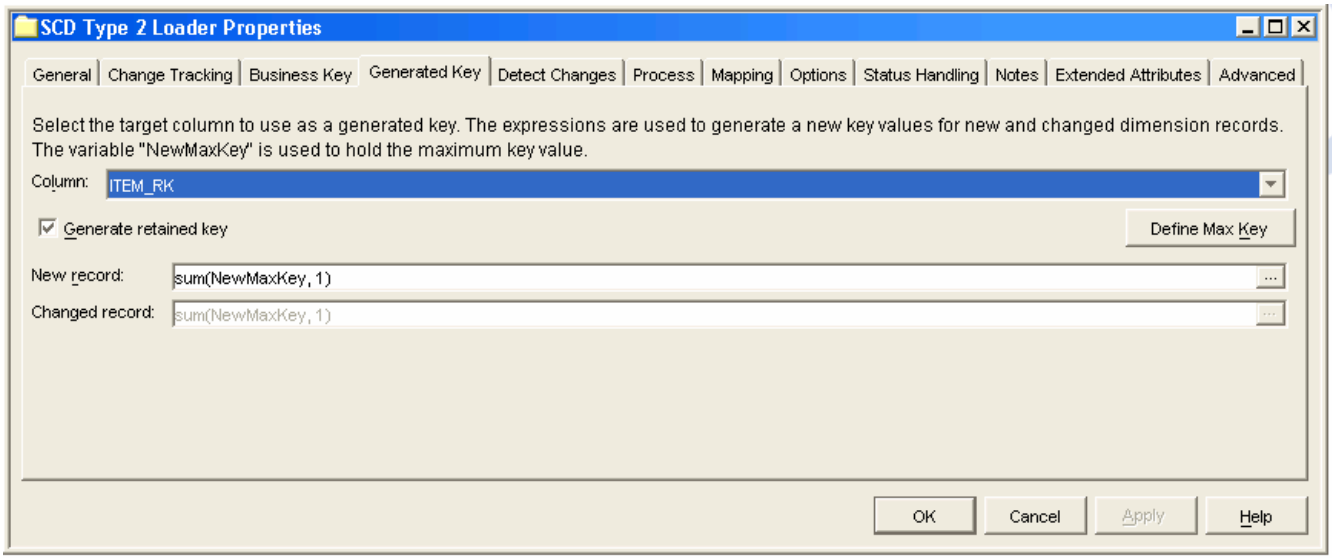


Figure 11. Adding a generated (retained) key allows us to uniquely identify any value in the table.

In Figure 11, the retained key is defined. In this case, we are using a reference table to store unique key values for each table/key in use in our warehouse. The way this works is in the generation of the key. An algorithm is provided to look this up from a data table in which the next available key is kept for each key being retained. This algorithm also automatically updates the table with the next available key value after the current one is taken. The benefit of the retained (surrogate) key in this example is that we can ensure that we won't have duplicate primary keys in our tables since we're managing them ourselves. If we depended on the primary (business) keys coming from source systems, we might end up with duplicates through namespace collisions. If the overhead can be managed by a tool, rather than manually, there is little risk to using business and surrogate keys in this way. Since SAS 9 ETL Studio does this for you, it's a good practice to use.

The final step is to select the columns to be used to detect when a change has occurred. This can be done by looking at the values in all of the content values in the table being processed. Here, "content values" mean all of the columns except the business and generated key columns. In some cases, fewer values can be used if some of them are known to be truly non-changing.

The actual change detection is done by a checksum-like algorithm that computes a distinct value based on all the selected input columns. Then, these checksum-like values are actually compared to detect change. This is done to prevent large text-based compares having to be done against each row, which quickly gets prohibitively expensive as data volumes rise. That's not to say that the checksum-like algorithm is extremely fast. It itself has to scan all of the data in question to generate the checksum (based on the public-domain MD5 algorithm), so for very large tables, recomputing all such checksum-like values could take a very long time.

CONCLUSION

This paper has discussed three advanced warehousing topics that can be performed in the visual environment of SAS 9 ETL Studio. Taking these three advanced functions in conjunction with the basic functionality set in ETL Studio can allow for production-quality warehouses to be designed and implemented in SAS 9.

Planning is of course required to ensure that these advanced capabilities are used appropriately. However, its hard to do too much data validation, since problematic data that is found early leads to much greater quality in the final warehouse data. Treating exceptions such as data validation issues or other non-standard code runtime issues is essential, both as a baselining process as well as to treat true exceptional cases as they occur. The granularity with which problems need to be detected and handled needs to be thought of in the early flow design process.

Dealing with slowly-changing dimensional data is a key warehousing activity. Being able to set this up in a straightforward fashion in the warehousing design environment can make ETL designers far more efficient since the complex logic can be compartmentalized and therefore have a minimal impact on overall warehouse flow.

The reader will hopefully realize that any large scale data management problem can truly benefit from these types of capabilities that are often very difficult to achieve consistently and efficiently when writing code without the benefit of an integrated design environment.

REFERENCES

Kimball, Ralph (1996), "Slowly Changing Dimensions", *DBMS Magazine,* April issue.

Mehler, Gary (2003), "Next Generation Warehousing with Version 9", *Proceedings of 28th Annual SAS Users Group International Conference*.

CONTACT INFORMATION

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

Gary Mehler
SAS
SAS Campus Drive
Cary NC 17513
Work Phone: (919) 677-8000
Fax: (919) 677-4444
Email: gjm@sas.com
Web: www.sas.com

SAS and all other SAS Institute Inc. product or service names are registered trademarks or trademarks of SAS Institute Inc. in the USA and other countries. ® indicates USA registration.

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