

Barriers to Data Quality

Author: BALWANT RAI
Organization: Evaltech, Inc.
**Evaltech Research Group,
Data Warehousing Practice.**
Date: 06/18/04
Email: erg@evaltech.com



Abstract:

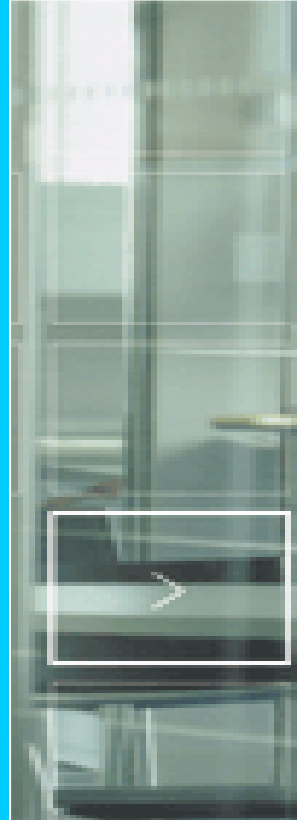
This article contains the information about the barriers to data quality. Two factors may hinder us from delivering data of the quality data warehouse users require:

- The legacy of disparate data maintained by source systems
- An organization's business policies toward data management.

This paper gives the information about the types of approaches that can be used for developing our operational systems, types of different models depending on each modeler's bias.

Intellectual Property / Copyright Material

All text and graphics found in this article are the property of the Evaltech, Inc. and cannot be used or duplicated without the express written permission of the corporation through the Office of Evaltech, Inc.



Data quality inside data warehouse

As new application data is added to the data warehouse environment, all the integration and transformation layer issues will be re-addressed, and the new data may also uncover more hidden anomalies and relationships even in the warehouse itself. However, another key reason is that the data warehouse contains data collected over a spectrum of time. In some cases, the spectrum is as long as ten years. The problem with data collected over time is that data itself changes over time. In some cases, the changes are slow and subtle. In other cases, the changes are fast and radical. In any case, it is simply a fact of life that data changes over time. And with these changes comes the need to integrate data over time within the data warehouse after it has already been loaded. Even if data is entered perfectly from the applications and integration and transformation programs, there will still be a need to examine data quality inside the data warehouse over time. It is hard that the data remained constant over those years.

For example, money is measured in the local currency in some year 'YYYY'. But in next year, money is measured in Eurodollars. Trying to perform a cash analysis between these years will be very difficult because the underlying meaning of the data has changed. Therefore, even if data quality has been perfected elsewhere, it remains to be perfected one more time after the data enters the warehouse simply because data ages inside the warehouse.

Two factors may hinder us from delivering data of the quality data warehouse users require:

- The legacy of disparate data maintained by source systems
- An organization's business policies toward data management.

Process-oriented: In this approach each major business function is treated as a self-contained black box and considering data a byproduct of the process. The process-oriented approach to applications development results in "encapsulated" data, which, for all practical purposes, remains hidden from other applications. Consequently, each application may organize its data to meet the specific needs of only those business processes it supports, resulting in specialized, non-integral, "stovepipe" applications. The ramifications of this strategy are making themselves apparent in operational data store and data warehouse applications that require integrated data views across application boundaries.

Advantages of process-oriented approach:

- It's much easier to control by drawing firm boundaries for an application.
- It involves people experts in the business subject-matter pertinent to the application's processes. The number of participants in the analysis, design, and user-acceptance phases of the life cycle is relatively low
- This approach met the needs of a time when our objective was to achieve enormous improvements in performance and throughput within the operational environment.

Why does the process-oriented approach result in a portfolio of applications encompassing disparate data?

- The diverse perspectives of subject-matter experts (SMEs)
- The different biases of modelers.

Each of the problem's components can be assigned to separate teams responsible for bringing it through the applications-development life cycle. As long as all teams recognize which data is to be shared and help develop common designs for that data, they shouldn't run into problems.

A bank, for example, may use different business units to perform payment processing of mortgages, credit cards, and car loans. Traditionally, IT has matched its applications to the organizational structure of a business, developing a separate application for each product. Consequently, when an organization holds analysis sessions to develop business requirements for a product, only SMEs for that product participate. Unless people with a cross-product view participate in these sessions and ensure that applications are designed consistently, the resulting models will probably be divergent and contribute to the legacy of disparate data. SMEs from different areas of the business provide their perspectives of the corporation's business rules, slanted toward specific requirements of the process they are performing. So while the application models represent the needs of the individual business products, they may unwittingly sacrifice the corporation's requirement for integrated data.

Different models will emerge depending on each modeler's bias. I know of modelers who create exact representations of business statements made by SMEs.

Literal model: The modeler will represent each product as a separate entity if SMEs give detailed descriptions of their products. This is a "literal" model, because it reflects the business rules as actually stated by the SMEs. The literal model is much easier to validate, since it is a purer representation of an organization's business rules. For the same reason, business stakeholders will more easily understand literal models than meta-level models.

Meta-level modeling: These models often include entities such as Product and Product Characteristic. For example Withdrawal and Drawdown are the subtypes of Debit. Meta level models rely on sample-instance tables to illustrate how business rules are represented. On the other hand, Meta level models often help management view its organization from a new perspective. They expose commonalities among business concepts in various business units that may otherwise have seemed distinct because of different terminologies.

The problem becomes more serious when SMEs assigned to the teams also come from different areas of the business. Even though two applications may consider the Product entity necessary to support their processes, there is only a small probability that their Product models will be identical. While model structures from different teams may occasionally be quite similar, the real "creativity" occurs in designing attributes. Good modelers and data analysts working in isolation from one another may create different, yet completely valid. With each of these variants, the model introduces another occurrence of disparate data. If an organization intends to share its data values easily across its business areas, meta-data (data designs) must first be shared across the application portfolio.

In object-oriented analysis and design, the application's boundaries are defined in terms of the business processes supported, and the object classes are defined within that context. Most designer's partition objects class methods with minimal overlap among interfacing applications. However, the object class attributes are designed based on methods specific to the individual applications. Consequently, while object modeling aids reuse *within* an application's boundaries, reuse across the application portfolio is far from a reality.

The level of data integration an organization can achieve and the effort required to populate a data warehouse's databases with quality data depends on the amount of disparate data the organization maintains. We may accomplish integration by linking together related objects. Consider a bank planning to implement a customer-driven marketing approach. Currently, bankers are assigned to accounts, and account numbers vary from product to product. Therefore,

a single customer may have to deal with multiple bank representatives, depending on what products he or she uses. With its new plan, the bank intends to implement a customer relationship-oriented sales strategy that requires account executives to work as a team to handle each customer's needs consistently. To accomplish this objective, the bank must be able to view the entire portfolio of services provided to an individual customer.

Perhaps the greatest barrier that the bank faces in compiling customer portfolios is its stovepipe application portfolio. Since the bank developed each product oriented application in a black-box manner, each application has a unique identifier for Customer, Product, Business Unit, and Employee entities, and each uses a different strategy for assigning these identifiers. Each identifier's format and domain may vary from one application to the next. In other words, there is no way to link a customer's accounts across applications because no common identifiers exist. Many organizations have attempted to derive customer portfolios by matching attributes such as name, address, and tax identifiers. But none of these techniques is foolproof. Some accounts may be incorrectly assigned to the same derived customer portfolio simply because they share the same name. In some cases, the bank will create duplicate portfolios for customers using different names and addresses on their various accounts. Knowledgeable staff (often the account executives) must correct these errors manually, wasting valuable time.

The confusion created by such inconsistent identifiers in operational systems isn't a new problem. I know of one company whose applications have implemented disparate domains for the business unit code: Some applications used an alphanumeric code, while others were fully numeric. Some applications allowed the business to define valid values, while others generated the values themselves. To minimize confusion among the business staff, the company now publishes a cross-reference table in its corporate phone book that lists the official business unit number to the code maintained by each application. This is just one example of how a business would have benefited greatly had its applications shared the same meta-data. Data warehouse designers should consider maintaining cross-references between identifiers assigned to integrated entity instances and those in the source application, as well as for indicative data from the source system. For example, if the tax identifier is the primary attribute used to match customer accounts, the name and address used for a customer's accounts may vary. A valuable audit trail is available if a "source system customer" table is implemented in the data warehouse, linking the source application's customer identifier, name, and address to the data warehouse's integrated customer entry.

Finally, disparate data drives up data warehouse implementation costs. With disparate data, your data warehouse analysts assume the role of "legacy archeologists," sifting through the data available from the source applications in search of those attributes- that interest data warehouse users. These analysts are forced to engage in what I call "meta-data mining." They know that the desired business facts exist in the source application, but don't know how they are represented.

The difficulty of the mining task depends on what type of "application roadmap" is available. Lucky prospectors will have access to a fully documented application model complete with business rule specifications (such as business definitions, valid domains, edit rules, and derivation rules). Those with less luck will find only program-level specifications available. In the worst case-which occurs all too often-no documentation for the source application is available. Analysts must be prepared to wade through program code to determine whether any of an application's data elements correspond to the data warehouse's attributes. The approach used will have a major impact on the time you'll need for data mapping. Given a well-documented, repository-based application, an experienced data analyst needs an average of 15 minutes per attribute to complete data mapping. The same analyst will require three to five hours per attribute if forced to extract the mapping and business rules from the application's code.

Another factor you must consider when sizing your data warehouse project is the time required to define and code translation rules for populating the warehouse. In some cases translation will be

straightforward. For example, in a simple code translation, the source application maintains "1" to represent the account status is "open," while the letter o may be used to represent the same thing in the data warehouse. However, some translation rules can be quite complex. Suppose data warehouse users want to know when a customer relationship (in other words, an account) was closed. Each product may have its own business rules for determining when an account closed. For example, a customer must explicitly close an account from which funds are available "on demand" (such as a checking account or a credit card.) Therefore, the close date for these products is usually maintained as an explicit data element. However, term accounts, such as Certificates of Deposit or loans with maturity dates (such as mortgages, or auto loans) are considered closed when the amount owed and all associated fees are zero. In this case, the actual close date may not be captured in the source system, and the data warehouse will have to assume that the account was closed during the month in which it detects zero balances. Finally, brokered accounts, which are opened when a bank serves as agent for a customer purchasing securities, are never considered closed, merely inactive. Here, the source application may only be able to provide the last date of trade. The bank must establish a business rule to determine how long such accounts must remain inactive before the data warehouse considers it closed. Obviously, the more complex these translation rules are, the more difficult they are to implement. Unfortunately, the extent of the data translation required to source the data warehouse is not evident until data mapping is completed. It is easy to underestimate data mapping and translation-rule specifications when developing initial data warehouse project estimates.