STONEBRIDGE

TECHNOLOGIES

Dealing with Missing Values In The Data Warehouse

Introduction

In an ideal world, every dimension key in each fact table record would point to a meaningful record in its respective dimension table. In addition, every dimension attribute of each dimension record would contain valid and meaningful information. In our real world, however, we often cannot assume that source data will be this complete. As a general rule, the more disparate and distinctly separate systems comprise the source for the warehouse, the more likely we will encounter issues with incomplete source data. Even when the warehouse is fed by a single centralized OLTP system, it is quite common to find that either or both of these two missing value problems must be reckoned with because missing values will undoubtedly affect the quality and usefulness of the warehouse.

To ignore the problem of missing dimension key values is to risk correct reporting and the confidence of data warehouse users. For example, the default behavior of some data transformation tools is to quietly skip over fact records with any missing dimension key values. Think about what that means in terms of data completeness. If a sales fact record does not contain a value for the salesperson dimension key, that sale record would simply not be loaded into the data warehouse. Do you really want your data warehouse users to see deflated sales measures just because sometimes the OLTP does not capture what sales person made a sale? If you ignore the possibility of missing dimension key values your data warehouse load process could be responsible for incorrect reports and measurements used for analysis.

To ignore the problem of missing dimension attribute values is to fail to add value to otherwise low quality data. This is particularly true in ROLAP systems, where missing values may appear as blank row headers on reports. Users will be less likely to get confused if meaningful descriptive labels show up when missing values are present in the data. For example, a “sales by customer income level” query might yield five different income levels with the amount of sales to customers in each level. What about the sixth category, sales to customers for which income level is not available or known? A user who sees this category appear on the report as “UNKNOWN” will clearly understand why these sales do not show up in any of the known categories. They will also be less likely to suspect that some type of reporting fluke has occurred than if the sales to this category showed up with a blank (i.e. NULL) label. Using default values for dimension attributes when loading the data warehouse can be implemented with varying degrees of sophistication depending on your user’s needs.

The first option for problem resolution should be to simply fix whatever is responsible for the incomplete source data, whether it be an extraction process problem or an unenforced business rule which fails to require users of the source OLTP system to enter certain values. For both the data warehouse team and the data warehouse users, eliminating the cause of the problem is always the ideal solution. Back in the real world, however, experience has revealed that when incomplete source data problems exist it is most often due to lack of business rules that would otherwise guarantee source values to be present. Unfortunately, missing or unenforced business rules in OLTP systems are typically one of the most difficult problems to correct. Examples of typical reasons for this difficulty are: technically inflexible mainframe legacy systems, practical issues where requiring entry of certain values would cause an operationally excessive burden on end users of the OLTP system, and unwillingness of OLTP end users to begin providing correct values for data fields.

Missing data value issues often must be dealt with even when your present OLTP system does not seem to contain fields with missing values. In many cases, a warehouse will be initialized with historical source data from backup tapes just before going into production. This old data will not contain fields added to the database structure after it was backed up to tape. In addition, at the time the data was backed up, certain fields may have been optional in the OLTP system which are now required. These are examples of how old OLTP data can have missing values even when the current system does not allow it. It is usually a practical impossibility to manually sift through all of this historical data to fill in the missing values.

Because of these issues associated missing values, it is important to have a useful philosophy and corresponding set of data warehouse loading techniques that specifically address the problem of missing values. Because the ideal solution of fixing the source data is often unobtainable, at least in the short term, this article delivers techniques and guidelines for dealing with missing values from the data warehouse loading perspective.

Missing Dimension Attribute Values

The difference between default, dummy, and missing values in the source data

Before tackling what to do about missing values, we must make sure we understand what they are and what they are not. There are several different types of data problems which seem to fall into this general category of poor data quality. These include default values, dummy values, and missing values. The first two are similar to each other, but are different than missing values.

Default values in the source system are values automatically provided for a user when they do not provide them. Loading procedures are not generally going to be aware of default values placed in the source data by OLTP applications. For example, the "New Customer" data entry screen on an OLTP system for a shipping company may simply default the customer type to "Commercial" since most customers will fall into that category. Of course the problem with default values is that sometimes the user does not bother to provide the correct value when the default one is not correct. For many data warehouses there is little or nothing the loading process can do to detect when default values were the correct values. So when a value exists, whether it happens to be a default one or not, the warehouse will simply load in the data as if all is well. In short, the problem of default OLTP values showing up in the data warehouse is not really a problem as long as the default values are correct frequently enough that data quality is not affected. If the default values are often incorrect you have a potentially significant data quality issue and will have to either devise a way to infer the correct value or, as already mentioned, go for the ideal solution and correct it in the OLTP system. Of course if you choose to devise a way to infer the correct value during the loading process the result must be a value that is believed to be more correct than what is stored in the OLTP system.

A dummy value is a value which is sometimes used *instead of* a missing value in the source system. For example, you'd never find a general ledger application that would allow a user to set up an account that didn't have an account type. But nonetheless users always find a way around software restrictions, so low and behold we see the "Miscellaneous" account type in the source data. The more often dummy values are used, the more it affects the quality of the data in the warehouse. Dimensional attribute fields containing dummy values pose essentially the same set of problems and potential solutions as described above for default values.

It should be apparent by now that default and dummy value problems in the warehouse do not leave a lot of options for solutions. This article focuses on handling one of the most common source data problems for data warehouses: missing values. This is the case where there is simply no value provided at all. Technically, the loading process either sees a NULL/empty value or one which only contains space characters. This situation is the easiest to detect from a technical standpoint because loading procedures can simply check for the absence of a value. What can be done when such missing values are detected during the load process is the focus of this paper.

What to do about missing dimension attribute values

What to do when missing dimension attribute values appear in the source data depends on several factors such as the relative importance of the attribute, the feasibility of correcting the source of the problem, and the feasibility of inferring the value while loading the warehouse.

Fix source data for controllable dimensions

Let's take a look at two common and important dimensions in many warehouses: product and customer. Although perhaps rare, the data administrator knows there will be product records which have no value for the **product_type** attribute in the product dimension. From a data administrative standpoint, the business should be in complete control of how it categorizes it's products, so it stands to reason that any products that are missing important attribute values like product type should be correctable (hence the term "controllable" dimension). Of course this means correcting the data *before* it gets to the warehouse. In this scenario you should strive to correct the source of the problem since it is both a very important attribute and is well within the control of the business.

Try creative inference for less controllable dimensions

For some attributes you may be able to devise a creative way to infer the value of an attribute when it is missing. During the loading of your customer dimension, you may encounter a new customer record which has no value for the **customer_type** attribute. When there is a new customer, it often coincides with initial purchases by that new customer. Your loading process could search the appropriate new fact records (i.e. sales) associated with this customer to find out what type of product they spent the most on. This, in turn, could be used to come up with a best guess for the most appropriate value of the **customer_type** attribute.

On the other hand, if it is known that there will be customer records which have no value for the **income_level** attribute in the customer dimension, there may be no real means for correcting the source of the problem. After all, some customers will simply refuse to reveal such information. What to do?

When all else fails, use a dummy value

When all other means for obtaining a meaningful value have failed, the warehouse loading process should use it's own dummy value. The most simplistic version of this involves converting all missing attribute values to a special value like the word **UNKNOWN** or **UNAVAILABLE**. This is far better than leaving the value NULL or full of space characters because it will be readily apparent to users that certain data is not known and the keepers of the warehouse *know* that it is unknown. This also helps ensure that when the attribute is used in a report as a row header or grouping (e.g. "sales by income level"), all of the sales to customers with unknown income levels end up in the **UNKNOWN** bucket. Technically, this should be the loading behavior even if some source customer records contain NULL and some contained space characters for the **income_level** attribute.

When one dummy isn't enough

For some warehouse designs, however, a more refined approach is needed. Let's assume that some of your customers are individual people and other customers are actual businesses. You customer dimension contains a **customer_type** attribute which indicates "Individual" or "Business". In this scenario you would likely have a customer dimension design where some attributes are specific to individuals, some are specific to businesses, but most apply to either type of customer. For example, the **income_level** attribute might only have meaning for individuals, while the **franchise_yn** attribute (designating whether or not the customer belongs to a franchise) would only have meaning for businesses. Attributes such as **customer name, country**, and **state** apply to both types of customers. What's the problem with converting a missing **income_level** field with the value **UNKNOWN** or **UNAVAILABLE**? For customers who are individuals this conversion seems appropriate, but what about customers who are businesses? Suppose a data warehouse user query asks for 1998 1st quarter sales by **income_level** for the **state** of Georgia. If the loading process has been replacing all missing **income_level** values with the value **UNKNOWN** for both individual *and* business customer types, the sales displayed in the **UNKNOWN** category may appear unusually large to a user. For one thing, this could mislead a user to believe or suspect that the quality of the **income_level** data is worse than it really is. As an example, if as much as 30% of the sales fall into this UNKNOWN category, it may be simply because 20% of the sales were to businesses (where the **income_level** is always missing) and the other 10% were to individuals which have no known value for **income_level**. It's bad enough to have to live with certain data quality issues in your warehouse, but if users believe the problem is worse than it really is then you may have a far worse problem: lack of sufficient utilization of the warehouse.

An equally important problem is somewhat the opposite of the former one: when the user accepts the data quality problem but fails to apply appropriate filters to get the most accurate data possible. The user simply accepts the fact that some individual customers will show up in the UNKNOWN income level category, but this acceptance combined with the blanket use of **UNKNOWN** for missing values in the loading process can actually mask problems with user's queries.

Borrowing from the previous example, if the user of the warehouse works for a company where only a small portion of sales come from businesses (say 4%), it will be easy to forget to exclude them when issuing queries which are intended to focus on the main customer base (i.e. the individuals). Using the same example query, 1998 1st quarter sales by **income_level** for the **state** of Georgia, suppose once again that the data warehouse loading process replaces

all missing **income_level** values with **UNKNOWN**, regardless of customer type. What if this query returned 7% of the sales in the **UNKNOWN** category? Notice that the user has not bothered to limit the results to customers with a **customer_type** of individual. Even if this particular user has a general idea of what the total 1998 1st quarter sales to "individual" customers in Georgia is supposed to be, they will not likely realize their mistake from the ever so slightly elevated total displayed by the query tool. Unless the user just luckily remembers to go back and filter out the business customers, they will not realize that some of the 7% in the **UNKNOWN** income level category are really there because those are business type customers which ought to be excluded in the first place.

So we see that if a user is comfortable with or at least not overly concerned with the amount of sales that end up in the **UNKNOWN** income level, this could actually allow users to unknowingly fail to apply a filter to their query which excludes the business customers. In this scenario, the user may believe they are looking at a total which reflects sales only for individual customers, when in fact some of the sales in the **UNKNOWN** income level category are business customers.

Use smart dummies for a more refined approach

We have seen just two examples of the basic shortcomings associated with using the same dummy replacement value for missing dimension attribute values regardless of context. There are likely many ways in which this type of problem can occur. However, it should be clear by now that to solve this set of problems, different dummy values should be used depending on the context. In our customer dimension example, since the **income_level** attribute only applies to customers with a customer type of *individual*, the data warehouse loading process should only replace missing **income_level** values with the dummy value **UNKNOWN** when the **customer_type** = '*individual*'. For *business* customers, the dummy value to use should be something simple like **NOT APPLICABLE**, or perhaps even more useful is a value such as **BUSINESS CUSTOMER**. For the users, using these "smart dummy" values helps draw the distinction between something that is not known and something that does not apply. As you might imagine, this extra level of customization can go a long way toward lessening the impact of poor data quality by making users more aware and informed through a slightly enhanced cleansing technique.

Missing Dimension Key Values

While loading a fact table, we generally expect each dimension key in each fact record to point to an existing record in it's associated dimension table. These dimension keys (or associated artificial dimension keys) are the foreign keys into their respective dimension tables. So far this description sounds just like foreign keys in an ER model and we're all comfortable with the way that works, right? In an OLTP system with an ER model, if a record needs to get inserted but there is a foreign key violation (i.e. no associated record in the parent table of the relationship) there is no mercy and the record is immediately denied insertion. This is both acceptable and desirable in the OLTP system because typically there is a user interacting with the system at the moment the record insertion is requested (e.g. adding a new customer or creating a new order). If there is something missing from the user's input and the foreign key constraint is being enforced (whether through the application, the database, or both), the user is there to fill in the missing information and must do so before being allowed to proceed. It is a great natural characteristic of OLTP systems that the data can be forced to be correct (or complete) before it is actually stored.

But let's consider storing data in the warehouse. Should the data warehouse be as strict as the OLTP systems that feed it when loading the fact table? I said that OLTP systems *can* (and usually do) force the user to provide complete information, but what about the cases where the source system does not or cannot strictly enforce even some of the more basic business rules that translate into foreign keys in the ER model? If it's a matter of *does not*, of course you should strive to remedy the OLTP system to enforce the foreign keys. On the other hand, there are cases where the source system *cannot* enforce foreign keys either by nature of their business or when it is simply a practical impossibility to do so.

Facing reality

In the retail industry, for example, retailers often do not know who their customers are. They ring up your items, you pay cash, and they know end up knowing little to nothing about you. The best they can do is to make some very

basic inferences about the type of customer that you are analyzing by the particular items you purchased. As you might imagine, then, a customer dimension for a warehouse containing retails sales will be fairly sparse. It will only contain customers that can actually be identified, such as those possessing merchant credit with the retailer or those who used a non-merchant credit card and their demographic data is obtained from a third party. Some retailers will ask for your phone number when making a cash purchase so that they may determine who you are and your demographics (i.e. your customer dimension attributes) from a third party source. Even those retailers cannot always find out who you are because not all of us are willing to give out our phone number. It is clear, then, that the OLTP system will not know whom the customer is for every single sale.

Your dimension needs a dummy

If the OLTP system does not always know who the customer is for every sale, then how is the data warehouse supposed to know what dimension key to assign when loading the sales fact table? Your data warehouse loading process must point such sale records to *something* in the customer dimension or there will be show-stopping issues for users of the warehouse. After all, if you just use a null value for the customer dimension key when such a sale record is inserted into the sales fact table then any query involving both the fact table and the customer dimension is subject to returning deflated results. That's because SQL will not join rows containing NULL values in the join columns. Once again, we must use our own dummy value so that the dimension key in the fact table points to a record in the customer dimension. Of course this dummy dimension key value must point to a dummy customer record which actually exists in the customer dimension.

The most simplistic implementation of this dummy record is perhaps the most common one. Simply assign the now infamous value of **UNKNOWN** to each attribute in this customer dimension record. If your company does know who the vast majority of their customers are, then this simple method should suffice because anything more sophisticated is probably not worth the effort required. In other words, if only 3% of your sales are associated with unknown customers then how much better off would your users really be if they *did* know their demographics?

What about dummy values for date and numeric data types?

Of course assigning labels like UNKNOWN and NOT APPLICABLE only work for columns with a character data type. What about dates and numbers? You can only put valid dates, valid numbers, and NULLs in those columns, so text is not allowed. Using special descriptive values for character columns is acceptable because we can make it very clear that the value is indeed a special value since the text is very readable and self explanatory (e.g. UNKNOWN, NOT APPLICABLE, etc).

However, this is not the case with dates and numbers because unless we use NULL, someone may knowingly or (more likely) unknowingly include unwanted dimension records when constraining on one of these columns. For example, if our sales rep dimension contains commission_rate as a numeric percent value and we use –999 as our special value to designate UNKNOWN, a query with a constraint on it such as commission_rate < .05 (e.g. commission rate less than 5%) will include sales rep dimension records which have –999 as the commission_rate value. Keep in mind that –999 would be used for sales reps for which we really don't know the commission rate, as opposed to assuming that it's zero when not given. The user probably does not wish for the above query to include such dimension records, but they may not even realize that they were included unless the query specifically displayed the commission rate value itself. The same kind of problem can occur with special dummy date values. If Dec. 31, 2500 is used as the dummy date value, someone looking at regional orders placed since Feb. 18, 1998 will see inflated order amounts because the query will include all orders for which the order date was not known because the order date Dec. 31, 2500 is greater than Feb. 18, 1998.

To avoid the pitfalls associated with using dummy values for date and numeric columns, it is best to leave them NULL when they are not known and cannot be estimated or derived. The tradeoff for avoiding these pitfalls is forcing your users to deal with empty cells and potentially empty row headers in their reports. The good news is that we are probably only talking about a small portion of the overall number of dimension attributes since most are the character type and in most designs there will not be a tremendous number of date and numeric columns in the dimensions anyway.

Some designs contain many columns of the numeric data type, but the most common explanation for this is the "code" fields (e.g. customer_type_code, product_type_code, etc). These are the codes associated with the descriptive fields (e.g. customer_type, product_type, etc). If the design requires the dimensions to capture both the codes and descriptions, there is no need to store the codes as numeric data types, even though they may be stored that way on the source system. The codes do not represent any quantitative value, so there is no benefit to storing them in the data warehouse as a numeric field. No one is going to sum up a code field in a query. Also, it is highly unlikely that anyone will need to apply a range constraint on a code field (e.g. product_type_code >= '12'). If you believe someone *would* need to apply a range constraint on such a code, you probably need to introduce a new attribute in the dimension that allows users to clearly and directly ask for what they want (e.g. add a product_group attribute which contains values such as "Group A" for product_type_code 1 through 5, "Group B" for product_type_code 6 through 11, "Group C" for product_type_code 12 and up, etc).

Why not simply reject

When your fact record source contains a record with a missing dimension key, many developers and architects will insist that "*This record is junk and therefore it does not belong in the warehouse!*" When this rationale is used to justify rejecting any such fact records from entering the warehouse, there is a significant risk. Specifically, if your loading process rejects some records because all of the dimension keys were not present, you may end up with a partially loaded warehouse from the user's perspective. For example, if one of our dimensions is SALES REP, but the OLTP system doesn't always capture who the sales representative is for every single sale, you will certainly not want to reject any sales from the warehouse just because the OLTP system did not know **who** made the sale. A sale is a sale is a sale, whether you know every last bit of information about it or not. If you reject a sale because you didn't know who sold it, the user of the warehouse will not be able to get correct totals at *any* level of summarization. If you accept a sale with an **UNKNOWN** sales rep, that sale can still be included in query results. When using a technique that accepts such sales records, the worst type of thing that can happen in terms of querying is that a "sales by sales rep" query will contain an amount that is not associated with any real sales person. At least the sales totals are correct, which is more than can be said for a warehouse that rejects some fact records.

In a nutshell, then, the message here is that we should think of loading the fact table as an all or nothing activity. Either accept all fact records or do not accept any at all. This is a general philosophy, however, for which exceptions can be legitimately made. For example, if sales transactions with zero dollar amounts appear in the source data and the data warehouse users contend that those fact records should not ever be included in the fact table, then perhaps you should specifically check for that condition before inserting into the fact table and decide whether to reject it or not. The other option would be to change the source extraction process so that *it* is responsible for deciding whether the transaction should be in the warehouse or not. By default, then, you should strive to load all fact records and only selectively reject for specific reasons that are driven by business goals. Blindly rejecting any fact record that is missing any dimension key values should not be a technical design assumption of your data warehouse loading processes.

The underlying principal: load all or load none

In justifying the acceptance of source fact records with missing dimension keys, my goal is to make the point that we generally should not use the same degree of strictness when loading the warehouse as when inserting and updating OLTP systems. Realize that I am intentionally ignoring the usual data cleansing associated with data integration (i.e. standardizing attribute values and dimension keys among disparate systems and multiple functional business areas). I am ignoring those issues because they are outside of the scope of this paper. What is being addressed here are the hiccups and anomalies in the OLTP data which must be tolerated when loading the data warehouse.

For example, suppose your sales warehouse is at the most granular level of detail possible and it is fed by the invoicing system that bills your customers. If your IS staff *and* users are certain that every single line item on every invoice should be associated with a certain sales representative, then you might consider it a serious problem if the sales rep dimension key is missing from a source fact record and abort the data warehouse load altogether with

appropriate error logging. On the other hand, if the source of your data ultimately comes from cash registers and you often (or even just sometimes) do not know who the customer was in the sale, then you should just accept those records with a missing customer dimension key by pointing them to the **UNKNOWN** record in the customer dimension. If you don't know who the customer is, you don't know who the customer is and no amount of clever cleansing and transforming is going to help you figure out who it was. If someone needs to be warned when you load a record with a missing source dimension key then just make sure that your loading process logs it in some way (e.g. appending to a log file or inserting a warning record into an error table).

When deciding what to tolerate and what not to tolerate during the load process, the line has to be drawn somewhere. So how do we make this decision? By default, you should assume that missing dimension keys and dimension attribute values will be technically acceptable and handled via the techniques described. If specific exceptions to this assumption need to be made for a specific fact table or dimension, then so be it. Try to make it the exception rather than the rule, however, because one of the most important data warehouse goals is to guarantee that users are working with a complete set of data and the fundamental means for achieving that is by loading all of the relevant data extracted from the source systems.

About the Author

John Hess is a Senior Data Warehouse Architect with StonebridgeTechnologies. He has served as the lead architect on several DW projects over the past 3 years, and enjoys mentoring other DW consultants within the organization. Prior to Stonebridge, John served in key roles as a UNIX developer, Oracle DBA, Informix DBA, and data modeler at a revenue management consulting firm for two years. John holds a bachelor's of science degree in computer engineering from Auburn University. John may be contacted at jhess@tacticsus.com.

-

For more information, please contact:

Stonebridge Technologies, Inc.
14800 Landmark Blvd.
Suite 250
Dallas, Texas 75240-7581
972.404.9755
800.776.9755
972.404.9754 Fax
www.sbti.com

