|  |
| --- |
|  |
|  |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| REVISION HISTORY | | | | | |
| Ver. | Description of Change | Author | Date | Approved | |
| Name | Effective Date |
| 1.0 | Initial status | [Elias Nema](mailto:Elias_Nema@epam.com) | 20-JAN-2014 |  |  |
|  |  |  |  |  |  |

Contents

[1. Delivering Dimension Tables 3](#_Toc384328927)

[1.1. The Basic Structure of a Dimension 3](#_Toc384328928)

[1.2. The Grain of a Dimension 4](#_Toc384328929)

[1.3. The Basic Load Plan for a Dimension 4](#_Toc384328930)

[2. Big Dimensions 5](#_Toc384328931)

[3. Small Dimensions 5](#_Toc384328932)

[4. Late-Arriving Dimension Records 5](#_Toc384328933)

[5. Calendar Date Dimension 6](#_Toc384328934)

[6. Multivalued Dimensions and Bridge Tables 7](#_Toc384328935)

[7. Ragged Hierarchies and Bridge Tables 8](#_Toc384328936)

[8. Source Books and Articles 9](#_Toc384328937)

# Delivering Dimension Tables

Dimension tables provide the context for fact tables and hence for all the measurements presented in the data warehouse. Although dimension tables are usually much smaller than fact tables, they are the heart and soul of the data warehouse because they provide entry points to data. We often say that a data warehouse is only as good as its dimensions. We think the main mission of the ETL team is the handoff of the dimension tables and the fact tables in the delivery step, leveraging the end user applications most effectively.

## The Basic Structure of a Dimension

All dimensions should be physically built to have the minimal set of components. The primary key is a single field containing a meaningless, unique integer. We call a meaningless integer key a surrogate. The data warehouse ETL process should always create and insert the surrogate keys. In other words, the data warehouse owns these keys and never lets another entity assign them. The primary key of a dimension is used to join to fact tables. Since all fact tables must preserve referential integrity, the primary dimension key is joined to a corresponding foreign key in the fact table. We get the best possible performance in most relational databases when all joins between dimension tables and fact tables are based on these single field integer joins. And finally, our fact tables are much more compact when the foreign key fields are simple integers.

All dimension tables should possess one or more other fields that compose the natural key of the dimension. An ID and designate the natural key field(s) with NK. The natural key is not a meaningless surrogate quantity but rather is based on one or more meaningful fields extracted from the source system. For instance, a simple static (nonchanging) employee dimension would probably have the familiar EMP ID field, which is probably the employee number assigned by the human resources production system. EMP\_ID would be the natural key of this employee dimension. We still insist on assigning a data warehouse surrogate key in this case, because we must insulate ourselves from weird administrative steps that an HR system might take. For instance, in the future we might have to merge in bizarrely formatted EMP\_IDs from another HR system in the event of an acquisition.

When a dimension is static and is not being updated for historical changes to individual rows, there is a 1-to-1 relationship between the primary surrogate key and the natural key. But we will see a little later in this chapter that when we allow a dimension to change slowly, we generate many primary surrogate keys for each natural key as we track the history of changes to the dimension. In other words, in a slowly changing dimension, the relationship between the primary surrogate key and the natural key is many-to-1. In our employee dimension example, each of the changing employee profile snapshots would have different and unique primary surrogate keys, but the profiles for a given employee would all have the same natural key (EMP\_ID). This logic is explained in detail in the section on slowly changing dimensions in this chapter.

The final component of all dimensions, besides the primary key and the natural key, is the set of descriptive attributes. Descriptive attributes are predominately textual, but numeric descriptive attributes are legitimate. The data warehouse architect probably will specify a very large number of descriptive attributes for dimensions like employee, customer, and product. Do not be alarmed if the design calls for 100 descriptive attributes in a dimension! Just hope that you have clean sources for all these attributes. More on this later.

The data warehouse architect should not call for numeric fields in a dimension that turn out to be periodically measured quantities. Such measured quantities are almost certainly facts, not descriptive attributes. All descriptive attributes should be truly static or should only change slowly and episodically. The distinction between a measured fact and a numeric descriptive attribute is not as difficult as it sounds. In 98 percent of the cases, the choice is immediately obvious. In the remaining two percent, pretty strong arguments can be made on both sides for modeling the quantity either as a fact or as a dimensional attribute. For instance, the standard (catalog) price of a product is a numeric quantity that takes on both roles. In the final analysis, it doesn’t matter which choice is made. The requesting applications will look different depending on where this numeric quantity is located, but the information content will be the same. The difference between these two choices will start to become important if it turns out that the standard price is actually slowly changing. As the pace of the change accelerates, modeling the numeric quantity as a measured fact becomes more attractive.

## The Grain of a Dimension

Dimensional modelers frequently refer to the grain of a dimension. By this they mean the definition of the key of the dimension, in business terms. It is then a challenge for the data warehouse architect and the ETL team to analyze a given data source and make sure that a particular set of fields in that source corresponds to the definition of the grain. A common and notorious example is the commercial customer dimension. It is easy to say that the grain of the dimension is the commercial customer. It is often quite another thing to be absolutely sure that a given source file always implements that grain with a certain set of fields. Data errors and subtleties in the business content of a source file can violate your initial assumptions about the grain. Certainly, a simple test of a source file to demonstrate that fields A, B, and C implement the key to the candidate dimension table source is the query:

Select A, B, C, count(\*)

From dimensiontablesource

Group by A, B, C

Having Count(\*) > 1

If this query returns any rows, the fields A, B, and C do not implement the key (and hence the grain) of this dimension table. Furthermore, this query is obviously useful, because it directs you to exactly the rows that violate your assumptions.

## The Basic Load Plan for a Dimension

A few dimensions are created entirely by the ETL system and have no real outside source. These are usually small lookup dimensions where an operational code is translated into words. In these cases, there is no real ETL processing. The little lookup dimension is simply created directly as a relational table in its final form.

But the important case is the dimension extracted from one or more outside sources. We have already described the four steps of the ETL data flow thread in some detail. Here are a few more thoughts relating to dimensions specifically.

Dimensional data for the big, complex dimensions like customer, supplier, or product is frequently extracted from multiple sources at different times. This requires special attention to recognizing the same dimensional entity across multiple source systems, resolving the conflicts in overlapping descriptions, and introducing updates to the entities at various points. These topics are handled in this chapter.

Data cleaning consists of all the steps required to clean and validate the data feeding a dimension and to apply known business rules to make the data consistent. For some simple, smaller dimensions, this module may be almost nonexistent. But for the big important dimensions like employee, customer, and product, the data-cleaning module is a very significant system with many subcomponents, including column validity enforcement, cross-column value checking, and row deduplication.

Data conforming consists of all the steps required to align the content of some or all of the fields in the dimension with fields in similar or identical dimensions in other parts of the data warehouse. For instance, if we have fact tables describing billing transactions and customer-support calls, they probably both have a customer dimension. In large enterprises, the original sources for these two customer dimensions could be quite different. In the worst case, there could be no guaranteed consistency between fields in the billing-customer dimension and the support-customer dimension. In all cases where the enterprise is committed to combining information across multiple sources, like billing and customer support, the conforming step is required to make some or all of the fields in the two customer dimensions share the same domains. We describe the detailed steps of conforming dimensions in the Chapter 4. After the conforming step has modified many of the important descriptive attributes in the dimension, the conformed data is staged again.

Finally, the data-delivering module consists of all the steps required to administer slowly changing dimensions (SCDs, described in this chapter) and write the dimension to disk as a physical table in the proper dimensional format with correct primary keys, correct natural keys, and final descriptive attributes. Creating and assigning the surrogate keys occur in this module. This table is definitely staged, since it is the object to be loaded into the presentation system of the data warehouse. The rest of this chapter describes the details of the data-delivering module in various situations.

# Big Dimensions

The most interesting dimensions in a data warehouse are the big, wide dimensions such as customer, product, or location. A big commercial customer dimension often has millions of records and a hundred or more fields in each record. A big individual customer record can have tens of millions of records. Occasionally, these individual customer records have dozens of fields, but more often these monster dimensions (for example, grocery store customers identified by a shopper ID) have only a few behaviorally generated attributes.

The really big dimensions almost always are derived from multiple sources. Customers may be created by one of several account management systems in a large enterprise. For example, in a bank, a customer could be created by the mortgage department, the credit card department, or the checking and savings department. If the bank wishes to create a single customer dimension for use by all departments, the separate original customer lists must be de-duplicated, conformed, and merged.

In the deduplication step, which is part of the data-cleaning module, each customer must be correctly identified across separate original data sources so that the total customer count is correct. A master natural key for the customer may have to be created by the data warehouse at this point. This would be a kind of enterprise-wide customer ID that would stay constant over time for any given customer.

In the conforming step, which is part of the data-conforming module, all attributes from the original sources that try to describe the same aspect of the customer need to be converted into single values used by all the departments. For example, a single set of address fields must be established for the customer. Finally, in the merge (survival) step, which is part of the delivery-module, all the remaining separate attributes from the individual source systems are unioned into one big, wide dimension record.

Later in this chapter, when we discuss slowly changing dimensions, we will see that the biggest dimensions are very sensitive to change, if it means that we generate new dimension records for each change. Hold that thought for a moment.

# Small Dimensions

Many of the dimensions in a data warehouse are tiny lookup tables with only a few records and one or two columns. For example, many transaction grained fact tables have a transaction type dimension that provides labels for each kind of transaction. These tables are often built by typing into a spreadsheet and loading the data directly into the final physical dimension table. The original source spreadsheet should be kept because in many cases new records such as new transaction types could be introduced into the business.

Although a little dimension like transaction type may appear in many different data marts, this dimension cannot and should not be conformed across the various fact tables. Transaction types are unique to each production system.

In some cases, little dimension tables that serve to decode operational values can be combined into a single larger dimension. This is strictly a tactical maneuver to reduce the number of foreign keys in a fact table. Some data sources have a dozen or more operational codes attached to fact table records, many of which have very low cardinalities. Even if there is no obvious correlation between the values of the operational codes, a single junk dimension can be created to bring all these little codes into one dimension and tidy up the design. The records in the junk dimension should probably be created as they are encountered in the data, rather than beforehand as the Cartesian product of all the separate codes. It is likely that the incrementally produced junk dimension is much smaller than the full Cartesian product of all the values of the codes. The next section extends this kind of junk-dimension reasoning to much larger examples, where the designer has to grapple with the problem of one dimension or two.

# Late-Arriving Dimension Records

Late-arriving data may need to be extracted via a different application or different constraints compared to normal contemporary data. Bad data obviously is picked up in the data-cleaning step.

A late-arriving dimension record presents a complex set of issues for the data warehouse. Suppose that we have a fictitious product called Zippy Cola. In the product dimension record for Zippy Cola 12-ounce cans, there is a formulation field that has always contained the value Formula A. We have a number of records for Zippy Cola 12-ounce cans because this is a Type 2 slowly changing dimension and other attributes like the package type and the subcategory for Zippy Cola 12-ounce cans have changed over the past year or two.

Today we are notified that on July 15, 2003 (a year ago) the formulation of Zippy Cola 12-ounce cans was changed to Formula B and has been Formula B ever since. We should have processed this change a year ago, but we failed to do so. Fixing the information in the data warehouse requires the following steps:

1. Insert a fresh new record with a new surrogate key for Zippy Cola 12-ounce cans into the Product dimension with the formulation field set to Formula B, the row effective datetime set to July 15, 2003, and the row end datetime set to the row effective datetime of the next record for Zippy Cola in the product dimension table. We also need to find the closest previous dimension record for Zippy Cola and set its row end datetime to the datetime of our newly inserted record. Whew!
2. Scan forward in the Product dimension table from July 15, 2003, finding any other records for Zippy Cola 12-ounce cans, and destructively overwrite the formulation field to Formula B in all such records.
3. Find all fact records involving Zippy Cola 12-ounce cans from July 15, 2003, to the first next change for that product in the dimension after July 15, 2003, and destructively change the Product foreign key in those fact records to be the new surrogate key created in Step 1.

There are some subtle issues here. First, we need to check to see if some other change took place for Zippy Cola 12-ounce cans on July 15, 2003. If so, we need only to perform Step 2.We don’t need a new dimension record in this special case.

In general, correcting bad data in the data warehouse can involve the same logic. Correcting a Type 1 field in a dimension is simplest because we just have to overwrite all instances of that field in all the records with the desired natural key. Of course, aggregate tables have to be recalculated if they have specifically been built on the affected attribute. Please see the aggregate updating section for fact tables in Chapter 6. Correcting a Type 2 field requires thoughtful consideration, since it is possible that the incorrect value has a specific time span.

# Calendar Date Dimension

Calendar date dimensions are attached to virtually every fact table to allow navigation of the fact table through familiar dates, months, ﬁscal periods, and special days on the calendar. You would never want to compute Easter in SQL, but rather want to look it up in the calendar date dimension. The calendar date dimension typically has many attributes describing characteristics such as week number, month name, ﬁscal period, and national holiday indicator. To facilitate partitioning, the primary key of a date dimension can be more meaningful, such as an integer representing YYYYMMDD, instead of a sequentially-assigned surrogate key. However, the date dimension table needs a special row to represent unknown or to-be-determined dates. If a smart date key is used, filtering and grouping should be based on the dimension table’s attributes, not the smart key.

When further precision is needed, a separate date/time stamp can be added to the fact table. The date/time stamp is not a foreign key to a dimension table, but rather is a standalone column. If business users constrain or group on time-of-day attributes, such as day part grouping or shift number, then you would add a separate time-of-day dimension foreign key to the fact table.

Here are some examples of columns that might be in a date dimension:

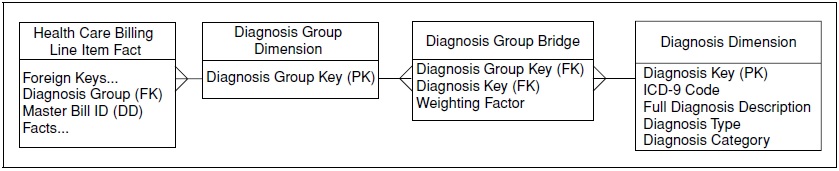
|  |  |  |
| --- | --- | --- |
| COLUMN | DATA TYPE | DESCRIPTION |
| DATE\_KEY | NUMBER / DATE | primary key; date in a format YYYYMMDD. If you are using Oracle there is no need to implement such a key and use DATE type instead, because of good internal representation of DATE in Oracle. |
| DAY\_OF\_WEEK\_NUMBER | NUMBER | number of a day in a week (from 1 to 7) |
| DAY\_OF\_WEEK\_DESC | VARCHAR2(25) | name of a day (Sunday, Monday, ... Saturday) |
| WEEKEND\_FLAG | NUMBER | flag whether a day is a weekend day (1) or not (0) |
| ISO\_WEEK\_NUMBER | NUMBER | number of an ISO week in a year (from 1 to 52) |
| DAY\_OF\_MONTH\_NUMBER | NUMBER | number of a day in a month (from 1 to 31) |
| MONTH\_VALUE | VARCHAR2(2) | 2-digits string in a format MM, number of a month in a year (from 01 to 12) |
| MONTH\_DESC | VARCHAR2(25) | full name of a month (January, February, ... December) |
| QUARTER\_VALUE | VARCHAR2(1) | 1-digit string in a format Q, number of a quarter in a year (from 1 to 4) |
| QUARTER\_DESC | VARCHAR2(2) | name of a quarter (Q1, Q2, Q3, Q4) |
| YEAR\_VALUE | VARCHAR2(4) | 4-digits string in a format YYYY (2011, 2012, ...) |

# Multivalued Dimensions and Bridge Tables

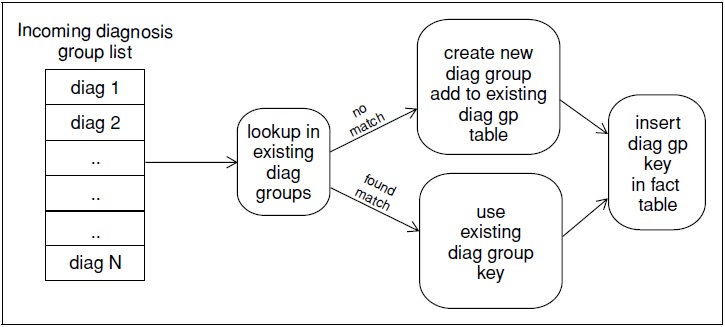
Occasionally a fact table must support a dimension that takes on multiple values at the lowest level of granularity of the fact table. Examples described in the other Toolkit books include multiple diagnoses at the time of a billable health care treatment and multiple account holders at the time of a single transaction against a bank account.

If the grain of the fact table is not changed, a multivalued dimension must be linked to the fact table through an associative entity called a bridge table.

To avoid a many-to-many join between the bridge table and the fact table, one must create a group entity related to the multivalued dimension. In the health care example, since the multivalued dimension is diagnosis, the group entity is diagnosis group. The diagnosis group becomes the actual normal dimension to the fact table, and the bridge table keeps track of the many-to-many relationship between diagnosis groups and diagnoses. In the bank account example, when an account activity record is linked to the multivalued customer dimension (because an account can have many customers), the group entity is the familiar account dimension. The challenge for the ETL team is building and maintaining the group entity table. In the health care example, as patient-treatment records are presented to the system, the ETL system has the choice of either making each patient’s set of diagnoses a unique diagnosis group or reusing diagnosis groups when an identical set of diagnoses reoccurs. There is no simple answer for this choice. In an outpatient setting, diagnosis groups would be simple, and many of the same ones would appear with different patients. In this case, reusing the diagnosis groups is probably the best choice. But in a hospital environment, the diagnosis groups are far more complex and may even be explicitly time varying. In this case, the diagnosis groups should probably be unique for each patient and each hospitalization. The admission and discharge flags are convenient attributes that allow the diagnosis profiles at the time of admission and discharge to be easily isolated.



**Figure 1 Using a bridge table to represent multiple diagnoses**

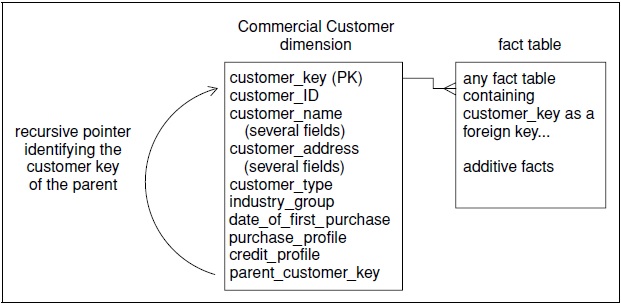


**Figure 2 Processing diagnosis groups in an outpatient setting**

# Ragged Hierarchies and Bridge Tables

Ragged hierarchies of indeterminate depth are an important topic in the data warehouse. Organization hierarchies are a prime example. A typical organization hierarchy is unbalanced and has no limits or rules on how deep it might be.

There are two main approaches to modeling a ragged hierarchy, and both have their pluses and minuses. We’ll discuss these tradeoffs in terms of the customer hierarchy shown in Figure 3.



**Figure 3 A customer dimension with a recursive pointer**

The recursive pointer approach:

(+) embeds the hierarchy relationships entirely in the customer dimension

(+) has simple administration for adding and moving portions of the hierarchy

(−) requires nonstandard SQL extensions for querying and may exhibit poor query performance when the dimension is joined to a large fact table

(−) can only represent simple trees where a customer can have only one parent (that is, disallowing shared ownership models)

(−) cannot support switching between different hierarchies

(−) is very sensitive to time-varying hierarchies because the entire customer dimension undergoes Type 2 changes when the hierarchy is changed

The hierarchy bridge table approach:

(+) isolates the hierarchy relationships in the bridge table, leaving the customer dimension unaffected

(+) is queried with standard SQL syntax using single queries that evaluate the whole hierarchy or designated portions of the hierarchy such as just the leaf nodes

(+) can be readily generalized to handle complex trees with shared ownership and repeating subassemblies

(+) allows instant switching between different hierarchies because the hierarchy information is entirely concentrated in the bridge table and the bridge table is chosen at query time

(+) can be readily generalized to handle time-varying Type 2 ragged hierarchies without affecting the primary customer dimension

(−) requires the generation of a separate record for each parent-child relationship in the tree, including second-level parents, third-level parents, and so on. Although the exact number of records is dependent on the structure of the tree, a rough rule of thumb is three times the number of records as nodes in the tree. Forty-three records are required in the bridge table to support the tree.

(−) involves more complex logic than the recursive pointer approach in order to add and move structure within the tree

(−) requires updating the bridge table when Type 2 changes take place within the customer dimension

# Source Books and Articles

1. Powell G. Oracle Data Warehouse Tuning for 10g. Oxford: Elsevier Digital Press, 2005.
2. Kimball R.,& Ross M. The Data Warehouse Toolkit: The Definitive Guide to Dimensional Modeling, Third Edition. Indianapolis: John Wiley & Sons, 2013.
3. Lane, P. Oracle Database Data Warehousing Guide, 11g Release 2 (11.2). Redwood City: Oracle, 2013.