Kimball Dimensional Modeling Techniques

KIMBALL GROUP KIMBALL UNIVERSITY

Table of Contents

Table of Contents continued...

Ralph Kimball introduced the data warehouse/business intelligence industry to dimensional modeling in 1996 with his seminal book, *The Data Warehouse Toolkit*. Since then, the Kimball Group has extended the portfolio of best practices. Drawn from *The Data Warehouse Toolkit, Third Edition* (coauthored by Ralph Kimball and Margy Ross, 2013), here are the “official” Kimball dimensional modeling techniques.

Fundamental Concepts

Gather Business Requirements and Data Realities

Before launching a dimensional modeling effort, the team needs to understand the needs of the business, as well as the realities of the underlying source data. You uncover the requirements via sessions with business representatives to understand their objectives based on key performance indicators, compelling business issues, decision-making processes, and supporting analytic needs. At the same time, data realities are uncovered by meeting with source system experts and doing high-level data profiling to assess data feasibilities.

Collaborative Dimensional Modeling Workshops

Dimensional models should be designed in collaboration with subject matter experts and data governance representatives from the business. The data modeler is in charge, but the model should unfold via a series of highly interactive workshops with business representatives. These workshops provide another opportunity to flesh out the requirements with the business. Dimensional models should not be designed in isolation by folks who don’t fully understand the business and their needs; collaboration is critical!

Four-Step Dimensional Design Process

The four key decisions made during the design of a dimensional model include:

1. Select the business process.
2. Declare the grain.
3. Identify the dimensions.
4. Identify the facts

The answers to these questions are determined by considering the needs of the business along with the realities of the underlying source data during the collaborative modeling sessions. Following the business process, grain, dimension, and fact declarations, the design team determines the table and column names, sample domain values, and business rules. Business data governance representatives must participate in this detailed design activity to ensure business buy-in.

Business Processes

Business processes are the operational activities performed by your organization, such as taking an order, processing an insurance claim, registering students for a class, or snapshotting every account each month. Business process events generate or capture performance metrics that translate into facts in a fact table. Most fact tables focus on the results of a single business process. Choosing the process is important because it defines a specific design target and allows the grain, dimensions, and facts to be declared. Each business process corresponds to a row in the enterprise data warehouse bus matrix.

Grain

Declaring the grain is the pivotal step in a dimensional design. The grain establishes exactly what a single fact table row represents. The grain declaration becomes a binding contract on the design. The grain must be declared before choosing dimensions or facts because every candidate dimension or fact must be consistent with the grain. This consistency enforces a uniformity on all dimensional designs that is critical to BI application performance and ease of use. Atomic grain refers to the lowest level at which data is captured by a given business process. We strongly encourage you to start by focusing on atomic-grained data because it withstands the assault of unpredictable user queries; rolled-up summary grains are important for performance tuning, but they pre-suppose the business's common questions. Each proposed fact table grain results in a separate physical table; different grains must not be mixed in the same fact table.

Dimensions for Descriptive Context

Dimensions provide the "who, what, where, when, why, and how" context surrounding a business process event. Dimension tables contain the descriptive attributes used by BI applications for filtering and grouping the facts. With the grain of a fact table firmly in mind, all the possible dimensions can be identified. Whenever possible, a dimension should be single valued when associated with a given fact row. Dimension tables are sometimes called the "soul" of the data warehouse because they contain the entry points and descriptive labels that enable the DW/BI system to be leveraged for business analysis. A disproportionate amount of effort is put into the data governance and development of dimension tables because they are the drivers of the user's BI experience.

Facts for Measurements

Facts are the measurements that result from a business process event and are almost always numeric. A single fact table row has a one-to-one relationship to a measurement event as described by the fact table's grain. Thus a fact table corresponds to a physical observable event, and not to the demands of a particular report. Within a fact table, only facts consistent with the declared grain are allowed. For example, in a retail sales transaction, the quantity of a product sold and its extended price are good facts, whereas the store manager's salary is disallowed.

Star Schemas and OLAP cubes

Star schemas are dimensional structures deployed in a relational database management system (RDBMS). They characteristically consist of fact tables linked to associated dimension tables via primary/foreign key relationships. An *online analytical processing (OLAP) cube* is a dimensional structure implemented in a multidimensional database; it can be equivalent in content to, or more often derived from, a relational star schema. An OLAP cube contains dimensional attributes and facts, but it is accessed through languages with more analytic capabilities than SQL, such as XMLA. OLAP cubes are included in this list of basic techniques because an OLAP cube is often the final step in the deployment of a dimensional DW/BI system, or may exist as an aggregate structure based on a more atomic relational star schema.

Grace Extensions to Dimensional Modeling

Dimensional models are resilient when data relationships change. All the following changes can be implemented without altering any existing BI query or application, and without any change in query results.

- Facts consistent with the grain of an existing fact table can be added by creating new columns.
- Dimensions can be added to an existing fact table by creating new foreign key columns, presuming they don't alter the fact table's grain.
- Attributes can be added to an existing dimension table by creating new columns.
- The grain of a fact table can be made more atomic by adding attributes to an existing dimension table, and then restating the fact table at the lower grain, being careful to preserve the existing column names in the fact and dimension tables.

Basic Fact Table Techniques

Fact Table Structure

A *fact table* contains the numeric measures produced by an operational measurement event in the real world. At the lowest grain, a fact table row corresponds to a measurement event and vice versa. Thus the fundamental design of a fact table is entirely based on a physical activity and is not influenced by the eventual reports that may be produced. In addition to numeric measures, a fact table always contains foreign keys for each of its associated dimensions, as well as optional degenerate dimension keys and date/time stamps. Fact tables are the primary target of computations and dynamic aggregations arising from queries.

Additive, Semi-Additive, and Non-Additive Facts

The numeric measures in a fact table fall into three categories. The most flexible and useful facts are fully *additive*; additive measures can be summed across any of the dimensions associated with the fact table. *Semi-additive* measures can be summed across some dimensions, but not all; balance amounts are common semi-additive facts because they are additive across all dimensions except time. Finally, some measures are completely *non-additive*, such as ratios. A good approach for non-additive facts is, where possible, to store the fully additive components of the non-additive measure and sum these components into the final answer set before calculating the final non-additive fact. This final calculation is often done in the BI layer or OLAP cube.

Nulls in Fact Tables

Null-valued measurements behave gracefully in fact tables. The aggregate functions (SUM, COUNT, MIN, MAX, and AVG) all do the “right thing” with null facts. However, nulls must be avoided in the fact table’s foreign keys because these nulls would automatically cause a referential integrity violation. Rather than a null foreign key, the associated dimension table must have a default row (and surrogate key) representing the unknown or not applicable condition.

Conformed Facts

If the same measurement appears in separate fact tables, care must be taken to make sure the technical definitions of the facts are identical if they are to be compared or computed together. If the separate fact definitions are consistent, *the conformed facts* should be identically named; but if they are incompatible, they should be differently named to alert the business users and BI applications.

Transaction Fact Tables

A row in a *transaction fact table* corresponds to a measurement event at a point in space and time. Atomic transaction grain fact tables are the most dimensional and expressive fact tables; this robust dimensionality enables the maximum slicing and dicing of transaction data. Transaction fact tables may be dense or sparse because rows exist only if measurements take place. These fact tables always contain a foreign key for each associated dimension, and optionally contain precise time stamps and degenerate dimension keys. The measured numeric facts must be consistent with the transaction grain.

Periodic Snapshot Fact Tables

A row in a *periodic snapshot fact table* summarizes many measurement events occurring over a standard period, such as a day, a week, or a month. The grain is the period, not the individual transaction. Periodic snapshot fact tables often contain many facts because any measurement event consistent with the fact table grain is permissible. These fact tables are uniformly dense in their foreign keys because even if no activity takes place during the period, a row is typically inserted in the fact table containing a zero or null for each fact.

Accumulating Snapshot Fact Tables

A row in an *accumulating snapshot fact table* summarizes the measurement events occurring at predictable steps between the beginning and the end of a process. Pipeline or workflow processes, such as order fulfillment or claim processing, that have a defined start point, standard intermediate steps, and defined end point can be modeled with this type of fact table. There is a date foreign key in the fact table for each critical milestone in the process. An individual row in an accumulating snapshot fact table, corresponding for instance to a line on an order, is initially inserted when the order line is created. As pipeline progress occurs, the accumulating fact table row is revisited and updated. This consistent updating of accumulating snapshot fact rows is unique among the three types of fact tables. In addition to the date foreign keys associated with each critical process step, accumulating snapshot fact tables contain foreign keys for other dimensions and optionally contain degenerate dimensions. They often include numeric lag measurements consistent with the grain, along with milestone completion counters.

Factless Fact Tables

Although most measurement events capture numerical results, it is possible that the event merely records a set of dimensional entities coming together at a moment in time. For example, an event of a student attending a class on a given day may not have a recorded numeric fact, but a fact row with foreign keys for calendar day, student, teacher, location, and class is well-defined. Likewise, customer communications are events, but there may be no associated metrics. *Factless fact tables* can also be used to analyze what didn't happen. These queries always have two parts: a factless coverage table that contains all the possibilities of events that might happen and an activity table that contains the events that did happen. When the activity is subtracted from the coverage, the result is the set of events that did not happen.

Aggregate Fact Tables or Cubes

Aggregate fact tables are simple numeric rollups of atomic fact table data built solely to accelerate query performance. These aggregate fact tables should be available to the BI layer at the same time as the atomic fact tables so that BI tools smoothly choose the appropriate aggregate level at query time. This process, known as *aggregate navigation*, must be *open* so that every report writer, query tool, and BI application harvests the same performance benefits. A properly designed set of aggregates should behave like database indexes, which accelerate query performance but are not encountered directly by the BI applications or business users. Aggregate fact tables contain foreign keys to shrunken conformed dimensions, as well as aggregated facts created by summing measures from more atomic fact tables. Finally, *aggregate OLAP cubes* with summarized measures are frequently built in the same way as relational aggregates, but the OLAP cubes are meant to be accessed directly by the business users.

Consolidated Fact Tables

It is often convenient to combine facts from multiple processes together into a single *consolidated fact table* if they can be expressed at the same grain. For example, sales actuals can be consolidated with sales forecasts in a single fact table to make the task of analyzing actuals versus forecasts simple and fast, as compared to assembling a drill-across application using separate fact tables. Consolidated fact tables add burden to the ETL processing, but ease the analytic burden on the BI applications. They should be considered for cross-process metrics that are frequently analyzed together.

Basic Dimension Table Techniques

Dimension Table Structure

Every dimension table has a single primary key column. This primary key is embedded as a foreign key in any associated fact table where the dimension row's descriptive context is exactly correct for that fact table row. Dimension tables are usually wide, flat denormalized tables with many low-cardinality text attributes. While operational codes and indicators can be treated as attributes, the most powerful dimension attributes are populated with verbose descriptions. Dimension table attributes are the primary target of constraints and grouping specifications from queries and BI applications. The descriptive labels on reports are typically dimension attribute domain values.

Dimension Surrogate Keys

A dimension table is designed with one column serving as a unique primary key. This primary key cannot be the operational system's natural key because there will be multiple dimension rows for that natural key when changes are tracked over time. In addition, natural keys for a dimension may be created by more than one source system, and these natural keys may be incompatible or poorly administered. The DW/BI system needs to claim control of the primary keys of all dimensions; rather than using explicit natural keys or natural keys with appended dates, you should create anonymous integer primary keys for every dimension. These *dimension surrogate keys* are simple integers, assigned in sequence, starting with the value 1, every time a new key is needed. The date dimension is exempt from the surrogate key rule; this highly predictable and stable dimension can use a more meaningful primary key.

Natural, Durable, and Supernatural Keys

Natural keys created by operational source systems are subject to business rules outside the control of the DW/BI system. For instance, an employee number (natural key) may be changed if the employee resigns and then is rehired. When the data warehouse wants to have a single key for that employee, a new *durable key* must be created that is persistent and does not change in this situation. This key is sometimes referred to as a *durable supernatural key*. The best durable keys have a format that is independent of the original business process and thus should be simple integers assigned in sequence beginning with 1. While multiple surrogate keys may be associated with an employee over time as their profile changes, the durable key never changes.

Drilling Down

Drilling down is the most fundamental way data is analyzed by business users. Drilling down simply means adding a row header to an existing query; the new row header is a dimension attribute appended to the GROUP BY expression in an SQL query. The attribute can come from any dimension attached to the fact table in the query. Drilling down does not require the definition of predetermined hierarchies or drill-down paths.

Degenerate Dimensions

Sometimes a dimension is defined that has no content except for its primary key. For example, when an invoice has multiple line items, the line item fact rows inherit all the descriptive dimension foreign keys of the invoice, and the invoice is left with no unique content. But the invoice number remains a valid dimension key for fact tables at the line item level. This *degenerate dimension* is placed in the fact table with the explicit acknowledgment that there is no associated dimension table. Degenerate dimensions are most common with transaction and accumulating snapshot fact tables.

Denormalized Flattened Dimensions

In general, dimensional designers must resist the normalization urges caused by years of operational database designs and instead denormalize the many-to-one fixed depth hierarchies into separate attributes on a flattened dimension row. Dimension denormalization supports dimensional modeling's twin objectives of simplicity and speed.

Multiple Hierarchies in Dimensions

Many dimensions contain more than one natural hierarchy. For example, calendar date dimensions may have a day to week to fiscal period hierarchy, as well as a day to month to year hierarchy. Location intensive dimensions may have multiple geographic hierarchies. In all of these cases, the separate hierarchies can gracefully coexist in the same dimension table.

Flags and Indicators as Textual Dimension Attributes

Cryptic abbreviations, true/false flags, and operational indicators should be supplemented in dimension tables with full text words that have meaning when independently viewed. Operational codes with embedded meaning within the code value should be broken down with each part of the code expanded into its own separate descriptive dimension attribute.

Null Attributes in Dimensions

Null-valued dimension attributes result when a given dimension row has not been fully populated, or when there are attributes that are not applicable to all the dimension's rows. In both cases, we recommend substituting a descriptive string, such as Unknown or Not Applicable in place of the null value. Nulls in dimension attributes should be avoided because different databases handle grouping and constraining on nulls inconsistently.

Calendar Date Dimensions

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. Filtering and grouping should be based on the date dimension'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.

Role-Playing Dimensions

A single physical dimension can be referenced multiple times in a fact table, with each reference linking to a logically distinct role for the dimension. For instance, a fact table can have several dates, each of which is represented by a foreign key to the date dimension. It is essential that each foreign key refers to a separate view of the date dimension so that the references are independent. These separate dimension views (with unique attribute column names) are called roles.

Junk Dimensions

Transactional business processes typically produce a number of miscellaneous, low-cardinality flags and indicators. Rather than making separate dimensions for each flag and attribute, you can create a single *junk dimension* combining them together. This dimension, frequently labeled as a *transaction profile dimension* in a schema, does not need to be the Cartesian product of all the attributes' possible values, but should only contain the combination of values that actually occur in the source data.

Snowflaked Dimensions

When a hierarchical relationship in a dimension table is normalized, low-cardinality attributes appear as secondary tables connected to the base dimension table by an attribute key. When this process is repeated with all the dimension table's hierarchies, a characteristic multilevel structure is created that is called a *snowflake*. Although the snowflake represents hierarchical data accurately, you should avoid snowflakes because it is difficult for business users to understand and navigate snowflakes. They can also negatively impact query performance. A flattened denormalized dimension table contains exactly the same information as a snowflaked dimension.

Outrigger Dimensions

A dimension can contain a reference to another dimension table. For instance, a bank account dimension can reference a separate dimension representing the date the account was opened. These secondary dimension references are called *outrigger dimensions*. Outrigger dimensions are permissible, but should be used sparingly. In most cases, the correlations between dimensions should be demoted to a fact table, where both dimensions are represented as separate foreign keys.

Integration via Conformed Dimensions

Conformed Dimensions

Dimension tables *conform* when attributes in separate dimension tables have the same column names and domain contents. Information from separate fact tables can be combined in a single report by using conformed dimension attributes that are associated with each fact table. When a conformed attribute is used as the row header (that is, the grouping column in the SQL query), the results from the separate fact tables can be aligned on the same rows in a drill-across report. This is the essence of integration in an enterprise DW/ BI system. *Conformed dimensions*, defined once in collaboration with the business's data governance representatives, are reused across fact tables; they deliver both analytic consistency and reduced future development costs because the wheel is not repeatedly re-created

Shrunken Rollup Dimensions

Shrunken dimensions are conformed dimensions that are a *subset* of rows and /or columns of a base dimension. *Shrunken rollup* dimensions are required when constructing aggregate fact tables. They are also necessary for business processes that naturally capture data at a higher level of granularity, such as a forecast by month and brand (instead of the more atomic date and product associated with sales data). Another case of conformed dimension subsetting occurs when two dimensions are at the same level of detail, but one represents only a subset of rows.

Drilling Across

Drilling across simply means making separate queries against two or more fact tables where the row headers of each query consist of identical conformed attributes. The answer sets from the two queries are aligned by performing a sort-merge operation on the common dimension attribute row headers. BI tool vendors refer to this functionality by various names, including stitch and multipass query.

Value Chain

A *value chain* identifies the natural flow of an organization's primary business processes. For example, a retailer's value chain may consist of purchasing to ware- housing to retail sales. A general ledger value chain may consist of budgeting to commitments to payments. Operational source systems typically produce transactions or snapshots at each step of the value chain. Because each process produces unique metrics at unique time intervals with unique granularity and dimensionality, each process typically spawns at least one atomic fact table.

Enterprise Data Warehouse Bus Architecture

The *enterprise data warehouse bus architecture* provides an incremental approach to building the enterprise DW/BI system. This architecture decomposes the DW/ BI planning process into manageable pieces by focusing on business processes, while delivering integration via standardized conformed dimensions that are reused across processes. It provides an architectural framework, while also decomposing the program to encourage manageable agile implementations corresponding to the rows on the enterprise data warehouse bus matrix. The bus architecture is technology and database platform independent; both relational and OLAP dimensional structures can participate.

Enterprise Data Warehouse Bus Matrix

The *enterprise data warehouse bus matrix* is the essential tool for designing and communicating the enterprise data warehouse bus architecture. The rows of the matrix are business processes and the columns are dimensions. The shaded cells of the matrix indicate whether a dimension is associated with a given business process. The design team scans each row to test whether a candidate dimension is well-defined for the business process and also scans each column to see where a dimension should be conformed across multiple business processes. Besides the technical design considerations, the bus matrix is used as input to prioritize DW/BI projects with business management as teams should implement one row of the matrix at a time.

The *detailed implementation bus matrix* is a more granular bus matrix where each business process row has been expanded to show specific fact tables or OLAP cubes. At this level of detail, the precise grain statement and list of facts can be documented.

Opportunity/Stakeholder Matrix

After the enterprise data warehouse bus matrix rows have been identified, you can draft a different matrix by replacing the dimension columns with business functions, such as marketing, sales, and finance, and then shading the matrix cells to indicate which business functions are interested in which business process rows. The *opportunity/stakeholder matrix* helps identify which business groups should be invited to the collaborative design sessions for each process-centric row.

Slowly Changing Dimension Techniques

Type 0: Retain Original

With slowly changing dimension *type 0*, the dimension attribute value never changes, so facts are always grouped by this original value. Type 0 is appropriate for any attribute labeled “original,” such as a customer’s original credit score or a durable identifier. It also applies to most attributes in a date dimension.

Type 1: Overwrite

With slowly changing dimension *type 1*, the old attribute value in the dimension row is overwritten with the new value; type 1 attributes always reflects the most recent assignment, and therefore this technique destroys history. Although this approach is easy to implement and does not create additional dimension rows, you must be careful that aggregate fact tables and OLAP cubes affected by this change are recomputed.

Type 2: Add New Row

Slowly changing dimension *type 2* changes add a new row in the dimension with the updated attribute values. This requires generalizing the primary key of the dimension beyond the natural or durable key because there will potentially be multiple rows describing each member. When a new row is created for a dimension member, a new primary surrogate key is assigned and used as a foreign key in all fact tables from the moment of the update until a subsequent change creates a new dimension key and updated dimension row. A minimum of three additional columns should be added to the dimension row with type 2 changes: 1) row effective date or date/time stamp; 2) row expiration date or date/time stamp; and 3) current row indicator.

Type 3: Add New Attribute

Slowly changing dimension *type 3* changes add a new attribute in the dimension to preserve the old attribute value; the new value overwrites the main attribute as in a type 1 change. This kind of type 3 change is sometimes called an alternate reality. A business user can group and filter fact data by either the current value or alternate reality. This slowly changing dimension technique is used relatively infrequently.

Type 4: Add Mini-Dimension

Slowly changing dimension *type 4* is used when a group of attributes in a dimension rapidly changes and is split off to a *mini-dimension.* This situation is sometimes called a rapidly *changing monster dimension*. Frequently used attributes in multimillion-row dimension tables are mini-dimension design candidates, even if they don’t frequently change. The type 4 mini-dimension requires its own unique primary key; the primary keys of both the base dimension and mini-dimension are captured in the associated fact tables.

Type 5: Add Mini-Dimension and Type 1 Outrigger

Slowly changing dimension *type 5* is used to accurately preserve historical attribute values, plus report historical facts according to current attribute values. Type 5 builds on the type 4 mini-dimension by also embedding a current type 1 reference to the mini-dimension in the base dimension. This enables the currently-assigned mini- dimension attributes to be accessed along with the others in the base dimension without linking through a fact table. Logically, you'd represent the base dimension and mini-dimension outrigger as a single table in the presentation area. The ETL team must overwrite this type 1 mini-dimension reference whenever the current mini-dimension assignment changes.

Type 6: Add Type 1 Attributes to Type 2 Dimension

Like type 5, slowly changing dimension *type 6* also delivers both historical and current dimension attribute values. Type 6 builds on the type 2 technique by also embedding current type 1 versions of the same attributes in the dimension row so that fact rows can be filtered or grouped by either the type 2 attribute value in effect when the measurement occurred or the attribute's current value. In this case, the type 1 attribute is systematically overwritten on all rows associated with a particular durable key whenever the attribute is updated.

Type 7: Dual Type 1 and Type 2 Dimensions

Slowly changing dimension *type 7* is the final hybrid technique used to support both as-was and as-is reporting. A fact table can be accessed through a dimension modeled both as a type 1 dimension showing only the most current attribute values, or as a type 2 dimension showing correct contemporary historical profiles. The same dimension table enables both perspectives. Both the durable key and primary surrogate key of the dimension are placed in the fact table. For the type 1 perspective, the current flag in the dimension is constrained to be current, and the fact table is joined via the durable key. For the type 2 perspective, the current flag is not constrained, and the fact table is joined via the surrogate primary key. These two perspectives would be deployed as separate views to the BI applications.

Dealing with Dimension Hierarchies

Fixed Depth Positional Hierarchies

A *fixed depth hierarchy* is a series of many-to-one relationships, such as product to brand to category to department. When a fixed depth hierarchy is defined and the hierarchy levels have agreed upon names, the hierarchy levels should appear as separate positional attributes in a dimension table. A fixed depth hierarchy is by far the easiest to understand and navigate as long as the above criteria are met. It also delivers predictable and fast query performance. When the hierarchy is not a series of many-to-one relationships or the number of levels varies such that the levels do not have agreed upon names, a ragged hierarchy technique must be used.

Slightly Ragged/Variable Depth Hierarchies

Slightly ragged hierarchies don't have a fixed number of levels, but the range in depth is small. Geographic hierarchies often range in depth from perhaps three levels to six levels. Rather than using the complex machinery for unpredictably variable hierarchies, you can force-fit slightly ragged hierarchies into a fixed depth positional design with separate dimension attributes for the maximum number of levels, and then populate the attribute value based on rules from the business.

Ragged/Variable Depth Hierarchies

Ragged hierarchies of indeterminate depth are difficult to model and query in a relational database. Although SQL extensions and OLAP access languages provide some support for recursive parent/child relationships, these approaches have limitations. With SQL extensions, alternative ragged hierarchies cannot be substituted at query time, shared ownership structures are not supported, and time varying ragged hierarchies are not supported. All these objections can be overcome in relational databases by modeling a ragged hierarchy with a specially constructed *bridge table*. This bridge table contains a row for every possible path in the ragged hierarchy and enables all forms of hierarchy traversal to be accomplished with standard SQL rather than using special language extensions.

The use of a bridge table for ragged variable depth hierarchies can be avoided by implementing a *pathstring attribute* in the dimension. For each row in the dimension, the pathstring attribute contains a specially encoded text string containing the complete path description from the supreme node of a hierarchy down to the node described by the particular dimension row. Many of the standard hierarchy analysis requests can then be handled by standard SQL, without resorting to SQL language extensions. However, the pathstring approach does not enable rapid substitution of alternative hierarchies or shared ownership hierarchies. The pathstring approach may also be vulnerable to structure changes in the ragged hierarchy that could force the entire hierarchy to be relabeled.

Advanced Fact Table Techniques

Fact Table Surrogate Keys

Surrogate keys are used to implement the primary keys of almost all dimension tables. In addition, single column surrogate fact keys can be useful, albeit not required. *Fact table surrogate keys*, which are not associated with any dimension, are assigned sequentially during the ETL load process and are used 1) as the single column primary key of the fact table; 2) to serve as an immediate identifier of a fact table row without navigating multiple dimensions for ETL purposes; 3) to allow an interrupted load process to either back out or resume; 4) to allow fact table update operations to be decomposed into less risky inserts plus deletes.

Centipede Fact Tables

Some designers create separate normalized dimensions for each level of a many-to- one hierarchy, such as a date dimension, month dimension, quarter dimension, and year dimension, and then include all these foreign keys in a fact table. This results in a *centipede fact table* with dozens of hierarchically related dimensions. Centipede fact tables should be avoided. All these fixed depth, many-to-one hierarchically related dimensions should be collapsed back to their unique lowest grains, such as the date for the example mentioned. Centipede fact tables also result when designers embed numerous foreign keys to individual low-cardinality dimension tables rather than creating a junk dimension.

Numeric Values as Attributes or Facts

Designers sometimes encounter numeric values that don’t clearly fall into either the fact or dimension attribute categories. A classic example is a product’s standard list price. If the numeric value is used primarily for calculation purposes, it likely belongs in the fact table. If a stable numeric value is used predominantly for filtering and grouping, it should be treated as a dimension attribute; the discrete numeric values can be supplemented with value band attributes (such as $0-50). In some cases, it is useful to model the numeric value as both a fact and dimension attribute, such as a quantitative on-time delivery metric and qualitative textual descriptor.

Lag/Duration Facts

Accumulating snapshot fact tables capture multiple process milestones, each with a date foreign key and possibly a date/time stamp. Business users often want to analyze the lags or durations between these milestones; sometimes these lags are just the differences between dates, but other times the lags are based on more complicated business rules. If there are dozens of steps in a pipeline, there could be hundreds of possible lags. Rather than forcing the user’s query to calculate each possible lag from the date/time stamps or date dimension foreign keys, just one time lag can be stored for each step measured against the process’s start point. Then every possible lag between two steps can be calculated as a simple subtraction between the two lags stored in the fact table.

Header/Line Fact Tables

Operational transaction systems often consist of a transaction header row that’s associated with multiple transaction lines. With *header/line* schemas (also known as *parent/child* schemas), all the header-level dimension foreign keys and degenerate dimensions should be included on the line-level fact table.

Allocated Facts

It is quite common in header/line transaction data to encounter facts of differing granularity, such as a header freight charge. You should strive to *allocate* the header facts down to the line level based on rules provided by the business, so the allocated facts can be sliced and rolled up by all the dimensions. In many cases, you can avoid creating a header-level fact table, unless this aggregation delivers query performance advantages.

Profit and Loss Fact Tables Using Allocations

Fact tables that expose the full equation of *profit* are among the most powerful deliverables of an enterprise DW/BI system. The equation of profit is (revenue) – (costs) = (profit). Fact tables ideally implement the profit equation at the grain of the atomic revenue transaction and contain many components of cost. Because these tables are at the atomic grain, numerous rollups are possible, including customer profitability, product profitability, promotion profitability, channel profitability, and others. However, these fact tables are difficult to build because the cost components must be allocated from their original sources to the fact table's grain. This allocation step is often a major ETL subsystem and is a politically charged step that requires high- level executive support. For these reasons, profit and loss fact tables are typically not tackled during the early implementation phases of a DW/BI program.

Multiple Currency Facts

Fact tables that record financial transactions in multiple currencies should contain a pair of columns for every financial fact in the row. One column contains the fact expressed in the true currency of the transaction, and the other contains the same fact expressed in a single standard currency that is used throughout the fact table. The standard currency value is created in an ETL process according to an approved business rule for currency conversion. This fact table also must have a currency dimension to identify the transaction's true currency.

Multiple Units of Measure Facts

Some business processes require facts to be stated simultaneously in several units of measure. For example, depending on the perspective of the business user, a supply chain may need to report the same facts as pallets, ship cases, retail cases, or individual scan units. If the fact table contains a large number of facts, each of which must be expressed in all units of measure, a convenient technique is to store the facts once in the table at an agreed standard unit of measure, but also simultaneously store conversion factors between the standard measure and all the others. This fact table could be deployed through views to each user constituency, using an appropriate selected conversion factor. The conversion factors must reside in the underlying fact table row to ensure the view calculation is simple and correct, while minimizing query complexity.

Year-to-Date Facts

Business users often request year-to-date (YTD) values in a fact table. It is hard to argue against a single request, but YTD requests can easily morph into "YTD at the close of the fiscal period" or "fiscal period to date." A more reliable, extensible way to handle these assorted requests is to calculate the YTD metrics in the BI applications or OLAP cube rather than storing YTD facts in the fact table.

Multipass SQL to Avoid Fact-to-Fact Table Joins

A BI application must never issue SQL that joins two fact tables together across the fact table's foreign keys. It is impossible to control the cardinality of the answer set of such a join in a relational database, and incorrect results will be returned to the BI tool. For instance, if two fact tables contain customer's product shipments and returns, these two fact tables must not be joined directly across the customer and product foreign keys. Instead, the technique of drilling across two fact tables should be used, where the answer sets from shipments and returns are separately created, and the results sort-merged on the common row header attribute values to produce the correct result.

Timespan Tracking in Fact Tables

There are three basic fact table grains: transaction, periodic snapshot, and accumulating snapshot. In isolated cases, it is useful to add a row effective date, row expiration date, and current row indicator to the fact table, much like you do with type 2 slowly changing dimensions, to capture a *timespan* when the fact row was effective. Although an unusual pattern, this pattern addresses scenarios such as slowly changing inventory balances where a frequent periodic snapshot would load identical rows with each snapshot.

Late Arriving Facts

A fact row is *late arriving* if the most current dimensional context for new fact rows does not match the incoming row. This happens when the fact row is delayed. In this case, the relevant dimensions must be searched to find the dimension keys that were effective when the late arriving measurement event occurred.

Advanced Dimension Table Techniques

Dimension-to-Dimension Table Joins

Dimensions can contain references to other dimensions. Although these relationships can be modeled with outrigger dimensions, in some cases, the existence of a foreign key to the outrigger dimension in the base dimension can result in explosive growth of the base dimension because type 2 changes in the outrigger force corresponding type 2 processing in the base dimension. This explosive growth can often be avoided if you demote the correlation between dimensions by placing the foreign key of the outrigger in the fact table rather than in the base dimension. This means the correlation between the dimensions can be discovered only by traversing the fact table, but this may be acceptable, especially if the fact table is a periodic snapshot where all the keys for all the dimensions are guaranteed to be present for each reporting period.

Multivalued Dimensions and Bridge Tables

In a classic dimensional schema, each dimension attached to a fact table has a single value consistent with the fact table's grain. But there are a number of situations in which a dimension is legitimately *multivalued*. For example, a patient receiving a healthcare treatment may have multiple simultaneous diagnoses. In these cases, the multivalued dimension must be attached to the fact table through a group dimension key to a bridge table with one row for each simultaneous diagnosis in a group.

A *multivalued bridge table* may need to be based on a type 2 slowly changing dimension. For example, the bridge table that implements the many-to-many relationship between bank accounts and individual customers usually must be based on type 2 account and customer dimensions. In this case, to prevent incorrect linkages between accounts and customers, the bridge table must include effective and expiration date/time stamps, and the requesting application must constrain the bridge table to a specific moment in time to produce a consistent snapshot.

Behavior Tag Time Series

Almost all text in a data warehouse is descriptive text in dimension tables. Data mining customer cluster analyses typically results in textual *behavior tags*, often identified on a periodic basis. In this case, the customers' behavior measurements over time become a sequence of these behavior tags; this time series should be stored as positional attributes in the customer dimension, along with an optional text string for the complete sequence of tags. The behavior tags are modeled in a positional design because the behavior tags are the target of complex simultaneous queries rather than numeric computations.

Behavior Study Groups

Complex customer behavior can sometimes be discovered only by running lengthy iterative analyses. In these cases, it is impractical to embed the behavior analyses inside every BI application that wants to constrain all the members of the customer dimension who exhibit the complex behavior. The results of the complex behavior analyses, however, can be captured in a simple table, called a *study group*, consisting only of the customers' durable keys. This static table can then be used as a kind of filter on any dimensional schema with a customer dimension by constraining the study group column to the customer dimension's durable key in the target schema at query time. Multiple study groups can be defined and derivative study groups can be created with intersections, unions, and set differences.

Aggregated Facts as Dimension Attributes

Business users are often interested in constraining the customer dimension based on aggregated performance metrics, such as filtering on all customers who spent over a certain dollar amount during last year or perhaps over the customer's lifetime. Selected *aggregated facts* can be placed in a dimension as targets for constraining and as row labels for reporting. The metrics are often presented as banded ranges in the dimension table. Dimension attributes representing aggregated performance metrics add burden to the ETL processing, but ease the analytic burden in the BI layer.

Dynamic Value Banding

A *dynamic value banding report* is organized as a series of report row headers that define a progressive set of varying-sized ranges of a target numeric fact. For instance, a common value banding report in a bank has many rows with labels such as "Balance from 0 to $10," "Balance from $10.01 to $25," and so on. This kind of report is dynamic because the specific row headers are defined at query time, not during the ETL processing. The row definitions can be implemented in a small value banding dimension table that is joined via greater-than/less-than joins to the fact table, or the definitions can exist only in an SQL CASE statement. The value banding dimension approach is probably higher performing, especially in a columnar database, because the CASE statement approach involves an almost unconstrained relation scan of the fact table.

Text Comments

Rather than treating freeform comments as textual metrics in a fact table, they should be stored outside the fact table in a separate comments dimension (or as attributes in a dimension with one row per transaction if the comments' cardinality matches the number of unique transactions) with a corresponding foreign key in the fact table.

Multiple Time Zones

To capture both universal standard time, as well as local times in *multi-time zone* applications, dual foreign keys should be placed in the affected fact tables that join to two role-playing date (and potentially time-of-day) dimension tables.

Measure Type Dimensions

Sometimes when a fact table has a long list of facts that is sparsely populated in any individual row, it is tempting to create a *measure type dimension* that collapses the fact table row down to a single generic fact identified by the measure type dimension. We generally do not recommend this approach. Although it removes all the empty fact columns, it multiplies the size of the fact table by the average number of occupied columns in each row, and it makes intra-column computations much more difficult. This technique is acceptable when the number of potential facts is extreme (in the hundreds), but less than a handful would be applicable to any given fact table row.

Step Dimensions

Sequential processes, such as web page events, normally have a separate row in a transaction fact table for each step in a process. To tell where the individual step fits into the overall session, a *step* *dimension* is used that shows what step number is represented by the current step and how many more steps were required to complete the session.

Hot Swappable Dimensions

Hot swappable dimensions are used when the same fact table is alternatively paired with different copies of the same dimension. For example, a single fact table containing stock ticker quotes could be simultaneously exposed to multiple separate investors, each of whom has unique and proprietary attributes assigned to different stocks.

Abstract Generic Dimensions

Some modelers are attracted to abstract generic dimensions. For example, their schemas include a single generic location dimension rather than embedded geographic attributes in the store, warehouse, and customer dimensions. Similarly, their person dimension includes rows for employees, customers, and vendor contacts because they are all human beings, regardless that significantly different attributes are collected for each type. Abstract generic dimensions should be avoided in dimensional models. The attribute sets associated with each type often differ. If the attributes are common, such as a geographic state, then they should be uniquely labeled to distinguish a store's state from a customer's. Finally, dumping all varieties of locations, people, or products into a single dimension invariably results in a larger dimension table. Data abstraction may be appropriate in the operational source system or ETL processing, but it negatively impacts query performance and legibility in the dimensional model.

Audit Dimensions

When a fact table row is created in the ETL back room, it is helpful to create an *audit dimension* containing the ETL processing metadata known at the time. A simple audit dimension row could contain one or more basic indicators of data quality, perhaps derived from examining an error event schema that records data quality violations encountered while processing the data. Other useful audit dimension attributes could include environment variables describing the versions of ETL code used to create the fact rows or the ETL process execution time stamps. These environment variables are especially useful for compliance and auditing purposes because they enable BI tools to drill down to determine which rows were created with what versions of the ETL software.

Late Arriving Dimensions

Sometimes the facts from an operational business process arrive minutes, hours, days, or weeks before the associated dimension context. For example, in a real-time data delivery situation, an inventory depletion row may arrive showing the natural key of a customer committing to purchase a particular product. In a real-time ETL system, this row must be posted to the BI layer, even if the identity of the customer or product cannot be immediately determined. In these cases, special dimension rows are created with the unresolved natural keys as attributes. Of course, these dimension rows must contain generic unknown values for most of the descriptive columns; presumably the proper dimensional context will follow from the source at a later time. When this dimensional context is eventually supplied, the placeholder dimension rows are updated with type 1 overwrites. Late arriving dimension data also occurs when retroactive changes are made to type 2 dimension attributes. In this case, a new row needs to be inserted in the dimension table, and then the associated fact rows must be restated.

Special Purpose Schemas

Supertype and Subtype Schemas for Heterogeneous Products

Financial services and other businesses frequently offer a wide variety of products in disparate lines of business. For example, a retail bank may offer dozens of types of accounts ranging from checking accounts to mortgages to business loans, but all are examples of an account. Attempts to build a single, consolidated fact table with the union of all possible facts, linked to dimension tables with all possible attributes of these divergent products, will fail because there can be hundreds of incompatible facts and attributes. The solution is to build a single *supertype fact table* that has the intersection of the facts from all the account types (along with a supertype dimension table containing the common attributes), and then systematically build separate fact tables (and associated dimension tables) for each of the subtypes. *Supertype* and *subtype* fact tables are also called *core* and *custom fact tables.*

Real-Time Fact Tables

Real-time fact tables need to be updated more frequently than the more traditional nightly batch process. There are many techniques for supporting this requirement, depending on the capabilities of the DBMS or OLAP cube used for final deployment to the BI reporting layer. For example, a "hot partition" can be defined on a fact table that is pinned in physical memory. Aggregations and indexes are deliberately not built on this partition. Other DBMSs or OLAP cubes may support deferred updating that allows existing queries to run to completion but then perform the updates.

Error Event Schemas

Managing data quality in a data warehouse requires a comprehensive system of data quality screens or filters that test the data as it flows from the source systems to the BI platform. When a data quality screen detects an error, this event is recorded in a special dimensional schema that is available only in the ETL back room. This schema consists of an error event fact table whose grain is the individual error event and an associated error event detail fact table whose grain is each column in each table that participates in an error event.