From ER Models to Dimensional Models:

Bridging the Gap between OLTP and OLAP Design, Part I

Daniel L. Moody and Mark A. R. Kortink

Daniel L. Moody is a Visiting Professor in the Department of Software Engineering, Charles University, Prague (visiting from Monash University, Melbourne, Australia).
moody@ksint.ms.mff.cuni.cz

Mark A. R. Kortink is the Managing Director of Kortink and Associates, an Australian–based IT management consultancy company.
mark@kortink.com

Dimensional modeling is a database design technique developed specifically for designing data warehouses. Its objectives are to create database structures that end users can easily understand and write queries against, and to optimize query performance. It has become the predominant approach to designing data warehouses in practice and has proven to be a major breakthrough in developing databases that can be used directly by end users. Dimensional modeling is not based on any theory, but has clearly been very successful in practice. This article, the first in a two-part series, examines the nature of dimensional modeling and proposes a possible explanation for why it has been so successful.

This article also explodes the popular myth that traditional ER modeling and dimensional modeling are fundamentally different and somehow incompatible. It shows that a dimensional model is just a restricted form of an ER model, and that there is a straightforward mapping between the two.

An ER model can be transformed into a set of dimensional models by a process of selective subsetting, denormalization, and (optional) summarization. Understanding the relationship between the two types of models can help to bridge the gap between operational system (OLTP) design and data warehouse (OLAP) design. It can also help to resolve the difficult problem of matching supply (operational data sources) and demand (end-user information needs) in data warehouse design. Finally, it results in a more complete dimensional design, which is less dependent on the designer's ability to choose the "right" dimensions.

Introduction

The data warehousing or OLAP (online analytical processing) environment is very different from the operational or OLTP (online transaction processing) environment, and therefore requires a different approach to database design (Kimball, 1995, 1996, 1997, 2002). There are two fundamental differences that suggest the need for different design approaches:

- **End-user access (visibility of database structure):** In a data warehousing environment, end users access data directly using user-friendly query and analysis (OLAP) tools (Glassey, 1998). This means users need to understand how data is structured in order to use the database effectively (though some of the details may be hidden by the OLAP tool). In operational systems, the database structure is effectively "hidden" by an application front end—users only access the database via data entry and enquiry screens.
- **Read-only access (query versus update performance):** In a data warehousing environment, users can retrieve data but cannot modify it, whereas in an operational environment, users make online updates to the database in response to business events. For most practical purposes, therefore, the data warehouse can be considered as a read-only database, as it is only updated via batch extract, transform, and load (ETL) processes.

Traditional database design techniques such as ER modeling and normalization are primarily designed for getting data efficiently and reliably into databases. However, structuring databases in this way also acts as a barrier to getting data out. The reason for this is that such methods tend to result in database structures which are too complex for end users to understand and write queries against. The average operational database consists of around a hundred tables, linked by a complex web of relationships, which most end users (not to mention most IT specialists) would find overwhelming. Even simple queries require multi-table joins, which are beyond the capabilities of most end users.

The primary cause of complexity in operational databases is the use of normalization. Normalization tends to multiply the number of tables, as it involves splitting out functionally dependent groups of attributes into separate tables. This minimizes data redundancy and maximizes update efficiency because each change can be made in a single place. It also maximizes data integrity by avoiding inconsistency and preventing insertion, deletion, and update anomalies (Codd, 1970). However, on the downside, it tends to penalize retrieval because of the number of joins required (Kent, 1983). Thus, normalization results in databases that are optimized for update rather than retrieval. Given that data warehouses are effectively read-only databases, normalization is not well suited for designing data warehouses.

Dimensional Modeling

Dimensional modeling, a database design technique developed specifically for designing data warehouses, addresses the limitations of traditional database design methods for such purposes. The approach was first defined by Kimball (1996), though he claims not to have invented it but to have merely documented what people had been doing in practice for many years. In particular, it was based on his observations of vendors in the business of providing data in a "user-friendly" form to their customers. The objectives of dimensional modeling are to:

- Produce database structures that are easy for end users to understand and write queries against.
- Optimize query performance (as opposed to update performance).

Dimensional modeling was a pivotal development in data warehousing, as it provided a viable means of

representing data in a way that end users could understand and write queries against. Effectively this transformed the dream of end users being able to satisfy their own information needs (the "self service" model of decision support as expounded by earlier data warehousing theorists) into reality. Dimensional modeling has become the predominant approach to designing data warehouses in practice and has proven to be a major breakthrough in designing database structures that can be understood and used directly by end users.

Star Schemas

Like most good ideas, dimensional modeling is very simple. It is based on a single, highly regular data structure called a star schema. This consists of a central table called the fact table, surrounded by a number of dimension tables.

The fact table forms the "center" of the star, while the dimension tables form the "points" of the star.

Fact tables typically correspond to particular types of business events—for example, orders, shipments, payments, bank transactions, airline reservations, and hospital admissions. Such tables usually contain numerical values (e.g., dollar amounts), which may be analyzed using statistical functions. Dimension tables provide the basis for analyzing data in the fact table—ways of "slicing and dicing" the data. Dimensions typically answer "who," "what," "when," "where," "how," and "why" questions about the business events stored in the fact table. A star schema may have any number of dimensions.

The terminology of "dimensional model" derives from the fact that a star schema may be visualized as a data "cube" where each dimension table represents a different spatial dimension (or more generally a hypercube, as a star schema may have any number of dimensions). As shown in Figure 2, each cell in the cube represents the intersection of values from each

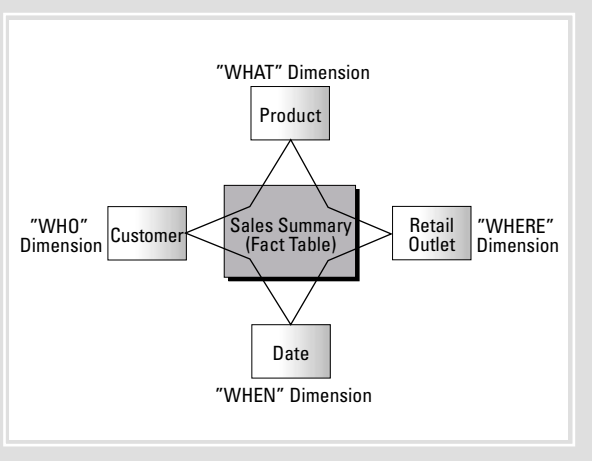

Figure 1. *Star Schema Example*

dimension. Of course, this metaphor loses its intuitive appeal once there are more than three dimensions, as most people have difficulty thinking in hyper-dimensional space!

Dimensions are often highly denormalized tables, typically consisting of one or more embedded hierarchies. For example, Customer, which forms a single dimension table in the star schema of Figure 1, consists of three independent hierarchies when they are normalized out (Figure 3). The hierarchical structure of dimensions provides the basis for analyzing data at different levels of detail—the ability to "drill down" and "roll up" in OLAP tools.

Snowflake Schemas

A snowflake schema is a star schema with fully normalized dimensions—it gets its name because it forms a shape similar to a snowflake. Rather than having a regular structure like a star schema, the "arms" of the snowflake can grow to arbitrary lengths in each direction. Figure 4 shows the equivalent snowflake schema for the star schema of Figure 1. While these two structures are equivalent in data content and the queries they support, the snowflake schema has a much more complex structure.

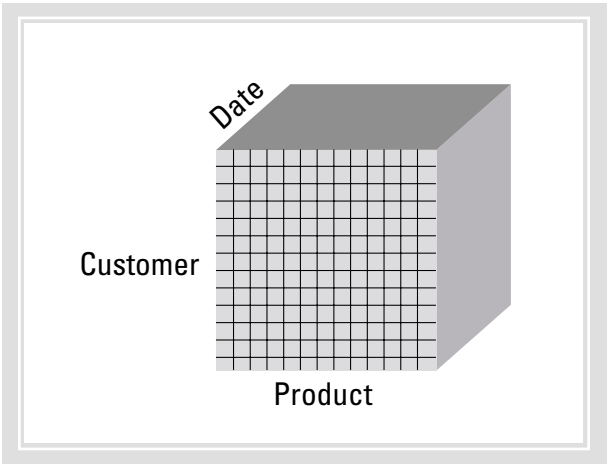

***Figure 2**. Data "Cube"*

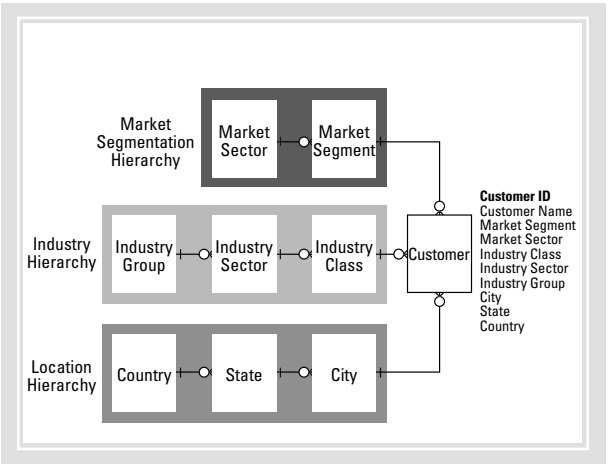

***Figure 3**. Embedded Hierarchies in Dimension Tables*

However, the advantage of the snowflake structure is that it explicitly shows the structure of each dimension rather than appearing as an unstructured collection of data items.

Kimball (1996, 2002) argues that "snowflaking" is undesirable because it adds unnecessary complexity, reduces query performance, and doesn't substantially reduce storage space. In the example, the snowflake schema has more than five times as many tables as the equivalent star schema. However, empirical tests show that the performance impact of snowflaking depends on the DBMS and/or OLAP tool used: some favor snowflakes while others favor star schemas (Spencer and Loukas, 1999). An advantage of the snowflake representation is that it explicitly shows the hierarchical structure of each dimension—in a star schema this requires tacit knowledge on the part of the user. While this may be overwhelming when presented all at once, it may be useful to allow end users to see the underlying structure of individual dimensions to understand how data can be sensibly

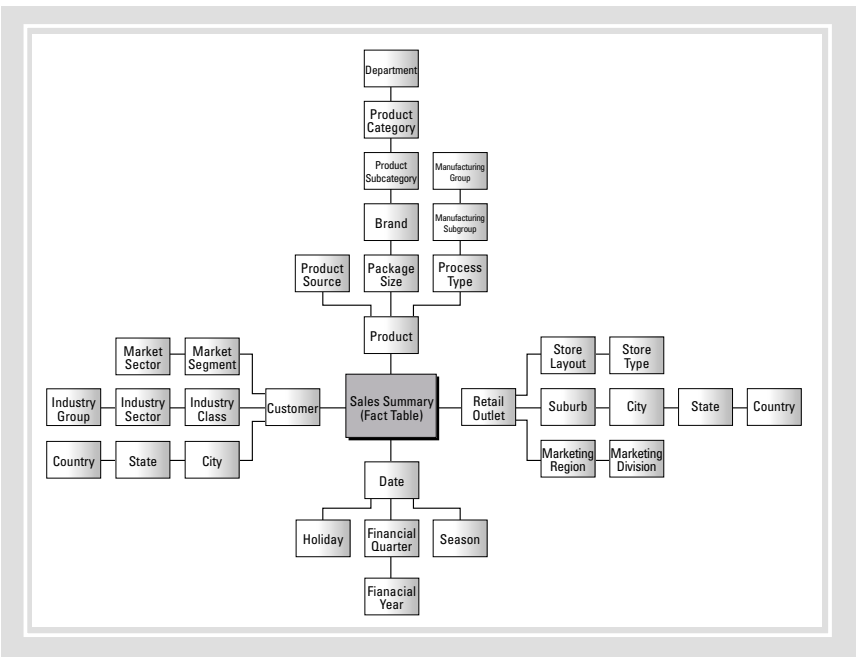

***Figure 4**. Snowflake Schema*

analyzed. This suggests that whether a snowflake or a star schema is used at the physical level, views should be constructed so that the user can see the structure of each dimension if required.

Strengths and Weaknesses of Dimensional Models

The major advantage of using dimensional models to represent data is that they drastically reduce the complexity of the database structure. This makes the database easier for people to understand and write queries against, by minimizing the number of tables and therefore the number of joins required. Dimensional models also optimize query performance. A star schema has a fixed structure that has no alternative join paths, which greatly simplifies the evaluation and optimization of queries (Raisinghani, 2000).

However, its strength is also its weakness. The fixed structure of the star schema limits the queries that can be written to the dimensions that have been defined. This means the designer must have a good idea in advance about the sort of questions users will want to ask (Craig, 1998). Star schemas make the most common queries easy to write but also restrict the way the data can be analyzed (Gallas, 1999). Another weakness of dimensional models is that not all information can be represented in dimensional form. Dimensional models assume an underlying hierarchical structure of data and exclude data that is naturally non-hierarchical (e.g., network structured data). Thus, dimensional modeling at best provides a partial solution (an "80/20" solution) to the problem of database query and analysis.

Theoretical Explanation: Why Dimensional Modeling Works

Dimensional modeling is not based on any theory, but has clearly been very successful in practice. It is a major contribution to the database design field—as important in practical terms as Codd's contribution of normalization (Codd, 1970) and Chen's contribution of ER modeling (Chen, 1976) and its influence on the practice of data warehouse design cannot be overstated. For these reasons, it is important to understand why it works. We offer a theoretical explanation for dimensional modeling's success in designing databases end users can understand and write queries against.

Principle #1: "Chunking"

Psychological studies show that due to limits on short-term memory, humans have a strictly limited capacity for processing information—this is estimated to be "the magical number seven, plus or minus two" concepts concurrently (Miller, 1956; Newell and Simon, 1972; Baddeley, 1994). If the amount of information received exceeds these limits, information overload ensues and comprehension degrades rapidly (Lipowski, 1975).

In one of the most important papers in the history of psychology, Miller (1956) described the results of a series of experiments using a wide range of different experimental stimuli (sounds, tastes, colors, points, numbers, and words). He observed a surprising similarity in "human channel capacity" across all these experiments, with a mean of 6.5 concepts, and one standard deviation, including 4 to 10 concepts (Figure 5). This has been described as the "inelastic limit of human capacity" (Uhr et al., 1962) or "cognitive bandwidth" (Moody, 2002), and represents one of the enduring laws of human cognition (Baddeley, 1994).

The primary mechanism used by the human mind to cope with large amounts of information is to organize it into "chunks" of manageable size (Weber, 1996; Baddeley, 1999; Miller, 1956). The ability to recursively develop information-saturated chunks is the key to people's ability to deal with complexity every day. The process of organizing data into a set of separate star schemas effectively provides a way of organizing a large amount of data into cognitively manageable "chunks." Although Kimball never refers

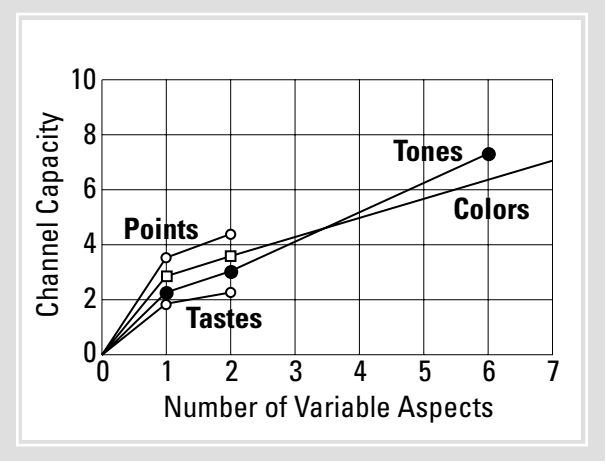

Figure 5. *Human Channel Capacity or Cognitive Bandwidth (Miller, 1956)*

to the "seven plus or minus two" principle, all the examples he uses in his original book (Kimball, 1996) implicitly obey this rule: an average of 6.6 entities per star schema, with a maximum of 11 and a minimum of 4. This corresponds remarkably well (in fact, almost perfectly) to the limits of human channel capacity defined by Miller!

Principle #2: Hierarchical Structuring

Hierarchy is one of the most common ways of organizing complexity for the purposes of human understanding, and is one of the basic principles of systems theory (Flood and Carson, 1993; Klir, 1985). Herbert Simon, the dual Nobel Prize winner, observed the use of hierarchy to organize complexity in organizational systems, social systems, biological systems, physical systems, and symbolic systems (Simon, 1996). Based on this, he proposed hierarchical organization as a general architecture for complexity. Hierarchical structures act as a complexity management mechanism by reducing the number of items one has to deal with at each level of the hierarchy. For example, instead of having to deal with a million customers, one can group them into a hundred market sectors and then into 10 market segments. Hierarchical structures are a familiar and natural way of organizing information, and are all around us in everyday life—for example, books are organized hierarchically, Web sites are organized hierarchically, the organizations we work in are organized hierarchically—even this article is organized hierarchically.

As discussed earlier, each dimension in a star schema typically consists of one or more hierarchies. These provide a way of classifying business events stored in the fact table, thereby reducing complexity. It is this hierarchical structure that provides the ability to analyze data at different levels of detail, and to "roll up" and "drill down" in OLAP tools. As we will see later, the process of building dimensional models is largely one of extracting hierarchical structures from enterprise data.

Implications for Practice

Together these principles provide a possible explanation for why dimensional modeling has been so successful in practice. Unlike traditional database design techniques, dimensional modeling is not based on any theory but simply evolved as a result of practice. However, closer examination shows that it is based on some general organizational principles (from psychology and systems theory) that have been used to manage complexity in a wide range of domains.

This is hardly surprising, as one of the most important problems in data warehousing is complexity management: how to present enormous volumes of data in a way that decision makers can understand. Current practice in dimensional modeling has more of the characteristics of an art than a science, and relies heavily on common-sense, experience, and "rules of thumb." Understanding the underlying principles on which dimensional models are based may help to develop principles for "good" dimensional modeling, which are largely lacking. For example, the "seven plus or minus two" principle could be used to define a limit on the number of dimensions in a star schema, to ensure that it represents a conceptually manageable "chunk" of data.

From ER Models to Dimensional Models: Exploding the Myths

There is a common misconception that dimensional modeling is fundamentally different from, and incompatible with, ER modeling. For example, as Kimball (1996) says:

> "Entity relation models are a disaster for querying because they cannot be understood by users and cannot be navigated usefully by DBMS software. Entity relation models cannot be used as the basis for enterprise data warehouses."

As a result, the starting point for most dimensional design approaches is that you must forget everything you ever learned about database design (Kimball, 1996, 1997, 1995, 2002; Oates, 2000). Sadly, this is a common theme in the introduction of any new technology, where proponents of the new technology claim that a whole new approach is needed, often when they do not fully understand the old approaches themselves. It is also part of good marketing to claim that something is totally new. We think it is important to dispel this notion, as it holds back the discipline of data warehouse design in two important ways:

- It acts as a barrier for people trained in traditional database design techniques to learn dimensional design. ER modeling and normalization have been used in practice to design database schemas for over two decades, and are taught as part of virtually every university level IT curriculum. Clearly, it would be useful to build on this wealth of knowledge and expertise than to simply discard it—for one thing, people learn better by building on what they already know.

- It acts as a barrier to establishing a "cumulative tradition" in the field, which is essential for scientific progress (Kuhn, 1970). Rather than simply saying that data warehousing is something totally new and different and trying to establish a new design discipline from its foundations, dimensional modeling should be linked to the existing body of knowledge in database design.

On closer examination, we find that a star schema is just a restricted form of an ER model (Figure 6):

- There is a single entity called the fact table, which is in third normal form (3NF). Violations to second normal form (2NF) would result in "double counting" in queries. The fact table forms an n-ary intersection entity (where n is the number of dimensions) between the dimension tables, and includes the keys of all dimension tables.
- There are one or more entities called dimension tables, each of which is related to the fact table via one or more one-to-many relationships. Dimension tables have simple keys, and are in at least 2NF. Transitive dependencies (3NF violations) are allowed, but partial dependencies (2NF violations) and repeating groups (1NF violations) are not.

As we will see, there is a straightforward transformation between a normalized ER model and a dimensional model, involving selective subsetting, denormalization, and (optional) summarization.

Deriving Dimensional Models from ER Models

Current approaches to dimensional design mostly take a top-down approach, in which dimensional models are developed directly from user query and analysis requirements ("demand side analysis"). While this may seem like a sound, "customer-driven" approach, it ignores one of the fundamental realities of the data warehouse environment. To a large extent, a data warehouse is simply a repackaging of

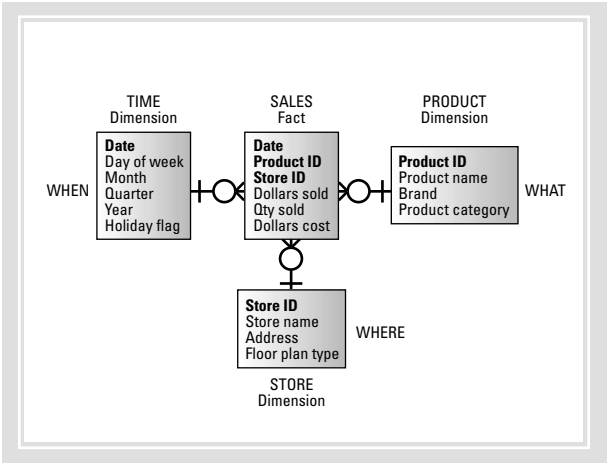

Figure 6. *ER Representation of a Star Schema*

operational data in a more accessible form. Data warehouse design is therefore a highly constrained design problem limited by what data is available from operational systems ("supply" issues).

In this article, we describe how to derive dimensional models from (normalized) ER models. This provides a way of repackaging operational data (usually stored in normalized tables) into a form that end users can understand and write queries against. This helps to resolve the difficult problem of matching "supply" (operational data sources) and "demand" (management information needs) in data warehouse design (Winter and Strauch, 2003). Deriving dimensional models from ER models also provides a more structured approach to dimensional design than starting from first principles. There is a large conceptual "leap" in getting from end-user analysis requirements to dimensional models, with which all but the most gifted designers have difficulty. Experience shows that there is a very steep learning curve involved and a great deal of experience required before people can successfully develop dimensional models on their own. The approach described in this paper can reduce this learning curve by providing a stepwise approach to developing dimensional models starting with the source data.

To illustrate the approach, we use an ER model for an order processing system (Figure 7). This consists of 42 entities, which is less than half the size of the average application data model, but large enough to illustrate the main principles of the approach. We will refer to this example throughout this article and the next (on advanced design issues, to appear in the Fall issue of the *Business Intelligence Journal*).

Even in a model of this size, the problem of complexity in normalized ER models is clearly evident. Most end users would find this schema incomprehensible, as it exceeds the limits of human cognitive capacity many times over. Even quite simple queries would require multi-table joins and/or subqueries, and as a result, end users would be heavily reliant on technical specialists to write queries for them.

The transformation of an ER Model to dimensional form takes place in four steps:

- Step 1: Classify entities
- Step 2: Design high-level star schema
- Step 3: Design detailed fact table
- Step 4: Design detailed dimension table

Step 1. Classify Entities

The first step in the transformation approach is to classify entities into three distinct categories:

- Transaction Entities: These entities record details of business events (e.g., orders, shipments, payments, insurance claims, bank transactions, hotel bookings, airline reservations, and hospital admissions). Most decision support applications focus on these events to identify patterns, trends, and potential problems and opportunities. The exception to this rule is the case of "snapshot" entities: entities recording a static level of some commodity at a point in time (e.g., account balances and

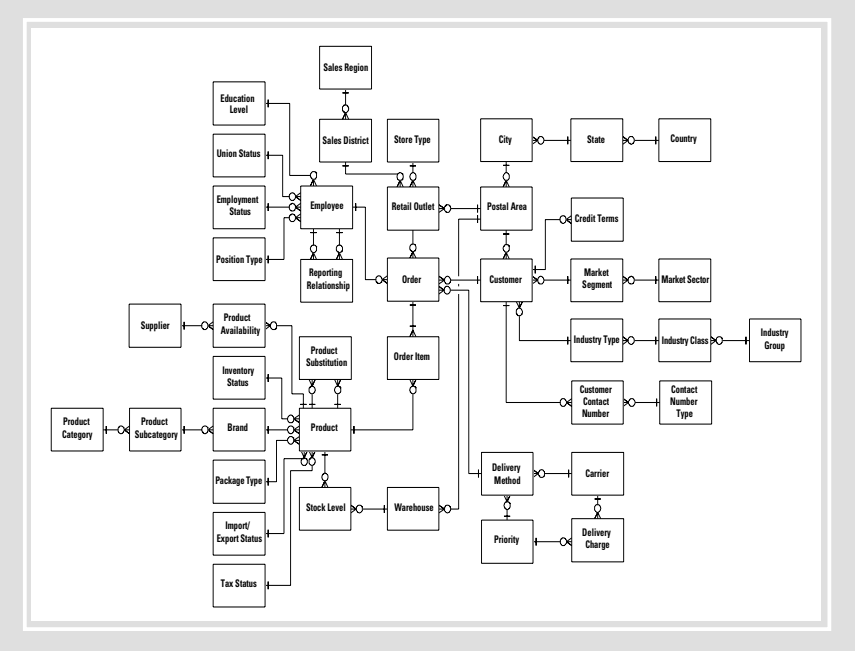

Figure 7. *ER Model for Retail Sales System*

There are six component entities: Customer, Employee, Retail Outlet, Product, Warehouse and Delivery Method. Product is a component of two different transactions (Stock Level and Order Item).

Finally, there are 31 classification entities defining separate but partially overlapping hierarchies in the model.

Note that some of the entities in the underlying data model do not fit into any of these categories. Such entities do not fit the hierarchical structure of a dimensional model and therefore cannot be represented in the form of a star schema (with some exceptions, to be discussed in the next issue of the *Business Intelligence Journal*). Thus, the process of "dimensionalizing" an ER model effectively "weeds out" non-hierarchical data.

inventory levels). These do not record business events as such, but the effect of events on the state of an entity.

- Component Entities: These entities are directly related to a transaction entity by a one-to-many relationship. They are involved in the business event, and answer "who," "what," "where," "how," and "why" questions about the event.
- Classification Entities: These entities are related to a component entity by a chain of one-to-many relationships. These define embedded hierarchies in the data model and are used to classify component entities.

The classification of entities for the example data model is shown in Figure 8. There are three transaction entities in the model: Order, Order Item, and Stock Level. Order and Order Item correspond to business events, while Stock Level records a static level (a "snapshot" entity). Order and Order Item represent different levels of detail of the same business event.

Step 2: High-Level Star Schema Design

In this step, relevant star schemas are identified and their high-level structures defined (table level design). At a high level, transaction entities correspond to fact tables and component entities correspond to dimension tables, but the mapping is not always one to one.

2.1 Identify Star Schemas Required

Each transaction entity is a candidate for a star schema. The process of creating star schemas is one of subsetting—dividing a large and complex model into manageable-sized chunks. Each star schema is centered on a single business event, and therefore represents a manageable-sized "chunk" of data. However, there is not always a one-to-one correspondence between transaction entities and star schemas:

- Not all transactions will be important for decision-making purposes: user input will be required to choose which transactions are relevant.
- Multiple star schemas at different levels of detail may be required for a particular transaction.
- When transaction entities are connected in a master-detail structure, they should be combined into a single star schema, as they represent different views of the same event. The split into "master" and "detail" is simply a requirement of normalization (1NF).

2.2 Define Level of Summarization

One of the most critical decisions in star schema design is to choose the appropriate level of granularity—the level of detail at which data is stored. In technical terms, granularity is defined as what each row in the fact table represents. At the top level, there are two main options in choosing the level of granularity:

- Unsummarized (transaction level granularity): this is the highest level of granularity where each fact table row corresponds to a single transaction or line item
- Summarized: transactions may be summarized by a subset of dimensions or dimensional attributes. In this case, each row in the fact table corresponds to multiple transactions

The lower the level of granularity (or conversely, the higher the level of summarization), the less storage space required and the faster queries will be executed. However, the downside is that summarization always loses information and therefore limits the types of analyses that can be carried out. Transaction-level granularity provides maximum flexibility for analysis, as no information is lost from the original normalized model.

A common misconception is that star schemas always contain summarized data—this is not necessarily the case and is not always desirable. However, for performance reasons—when dealing with large data volumes—or to simplify the view of data for a particular set of users, some level of summarization may be necessary. In practice, a range of star schemas at different levels of granularity are generally created for each transaction entity to suit the needs of different users. An important design consideration is that all of these star schemas should use exactly the same dimension tables or views defined on these tables—these are called "conforming" dimensions (Kimball, 1996, 2002). This ensures that users can "drill across" from one star

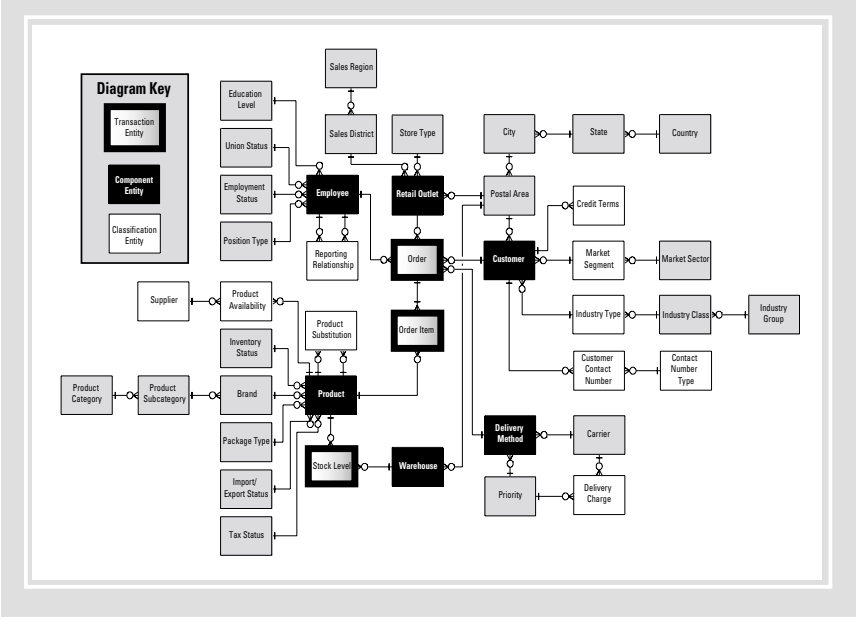

Figure 8. *Classification of Entities*

schema to another. Facts should also be defined consistently between different star schemas, to avoid inconsistent results from queries.

There is an almost infinite range of possibilities for creating summary level star schemas for a given transaction entity. In general, any combination of dimensions or dimensional attributes may be used to summarize transactions. For example, orders could be summarized by:

- Month (an attribute of the Date dimension) and Retail Outlet: this gives monthly sales totals for each retail outlet.
- Date, Product, and City (an attribute of the Retail Outlet dimension): this gives daily product sales for each city.

These specify the "group by" conditions for star schema load processes.

2.3 Identify Relevant Dimensions

The component entities associated with each transaction entity represent candidate dimensions for the star schema. However, the correspondence between component entities and dimensions is not always one to one:

- Not all component entities may be relevant for purposes of analysis or for the level of granularity chosen.
- If a component entity has no dependent attributes, its key will be included in the fact table but it will not have its own dimension table. This is called a degenerate dimension.
- Explicit dimensions are required to represent dates and times. Date and/or Time appear as explicit dimensions in most star schemas to support different types of historical analyses. These are not normally represented as entities in operational systems, as they are handled by built-in DBMS functions. These correspond to data types rather than entities at the operational level.

Figure 9 shows the transaction level star schema for the Order/Order Item transaction. Each row in the fact table corresponds to an individual Order Item (line item granularity).

The star schema has six dimension tables corresponding to the five component entities associated with the transaction, plus an additional Date dimension. There are multiple relationships to the Date dimension corresponding to the date the order was made and when it was posted to the general ledger. Order forms a degenerate dimension. If the time of orders is important for analysis, an additional Time dimension will be required. Date and Time are usually represented as separate dimensions to reduce the size of the dimension tables (Kimball, 1996).

As discussed earlier, transaction entities connected in a master-detail structure should be combined into a single star schema. Component entities associated with either of the entities become dimensions in the resulting star schema. In most cases, the "master" entity will become a "degenerate" dimension in the star schema: it forms part of the key of the fact table but has no corresponding dimension table (Figure 10). The reason for this is that all attributes of the

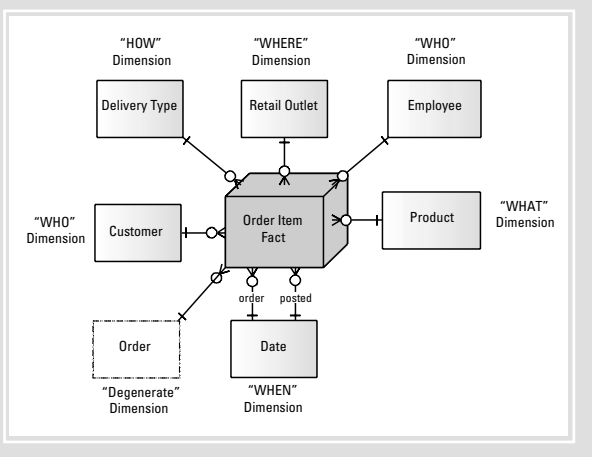

Figure 9. *Transaction Level Star Schema for Order/Order Item Transaction Entities*

master entity should be allocated down to the item level and stored in the fact table.

Figure 11 shows the transaction-level star schema for the Stock Level transaction entity, which is an example of a "snapshot" or level entity. Each row in the fact table records the inventory level of a particular product in a particular warehouse on a particular date. The star schema has three dimension tables, corresponding to the two component entities associated with the transaction entity plus an explicit Date dimension.

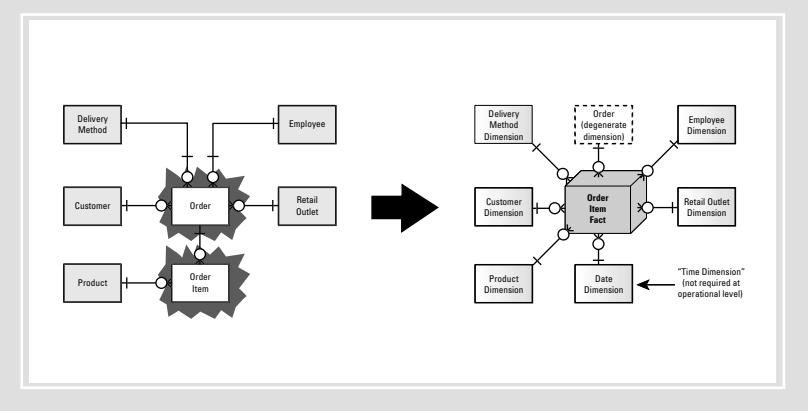

Figure 10. *Merging of "Master-Detail" Entities*

When non-transaction level granularity is chosen, the dimensions required will be determined by how the transactions are summarized. This will generally be a subset of the dimensions used in the transaction level star schema, though dimensions may themselves be subsets of the transaction level dimension tables (e.g., Month instead of Date). Figure 12 shows a summary level star schema in which daily orders are summarized by retail outlet and product. This requires three dimension tables: Date, Retail Outlet, and Product.

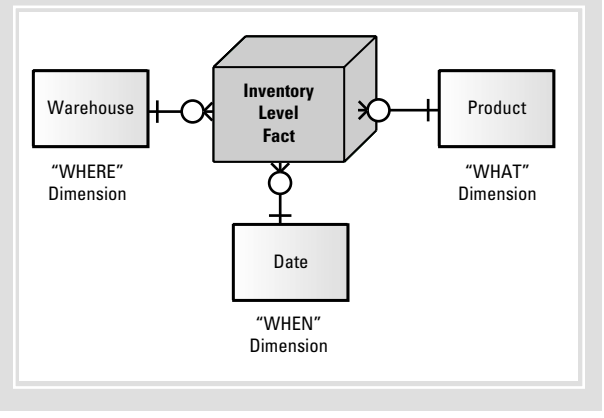

Figure 11. *Transaction Level Star Schema for Stock Level Transaction Entity*

Step 3: Detailed Fact Table Design

In this step, we complete the detailed design for fact tables in each star schema (attribute level design).

3.1 Define Key

The key of each fact table is a composite key consisting of the keys of all dimension tables plus any degenerate dimensions. This is different from how the key is defined for a normal relational table. In a relational database, the key should be minimal: no subset of the attributes in the key should also be unique. However, the key of a fact table includes all dimensional keys, and will therefore not be minimal in most cases. For example, in Figure 13 (the fact table for the Order transaction), the key consists of eight attributes, but only two (Order Number and Product ID) are necessary for uniqueness.

3.2 Define Facts

The non-key attributes of the fact table are measures (facts) that can be analyzed using numerical functions. What facts are defined depends on what event information is collected by operational systems—that is, what attributes are stored in transaction entities. However, a key concept in defining

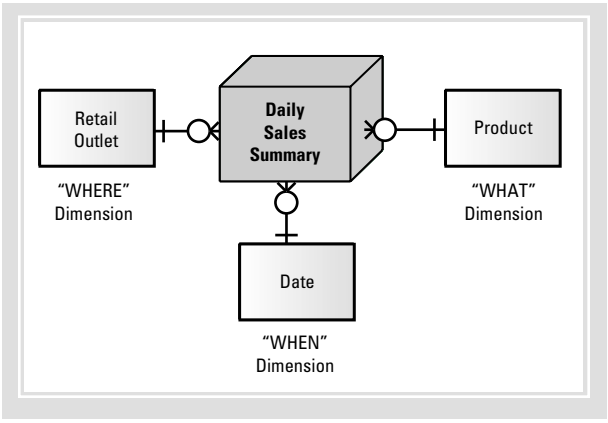

Figure 12. *Summary Star Schema for Order Transaction*

facts is that of additivity. Kimball (1996) defines three levels of additivity:

- Fully additive facts: these are facts that can be meaningfully added across all dimensions. For example, Quantity Ordered in the Order Item entity can be added across dates, products, or customers to get total sales volumes for a particular day, product, or customer.
- Semi-additive facts: these are facts that can be meaningfully added across some dimensions but not others (usually time). For example, Quantity On Hand from the Stock Level entity can be added across products and warehouses (to get the total quantity on hand for a particular product or warehouse) but not across time (as this would lead to double counting of stock). Quantity On Hand can be averaged over time (to get average daily inventory levels) but not added.
- Non-additive facts: these are facts that cannot be meaningfully added across any dimensions. For example, Unit Price from the Order Item entity cannot be added across any dimensions. Unit Price cannot even be sensibly averaged over time, because to calculate the average selling price of a product you need to take into account the quantity sold.

Wherever possible, additive facts should be used to prevent errors in queries. This means converting semi- and non-additive facts to additive facts. In the example, Unit Price (a non-additive fact) can be converted to an additive fact by multiplying it by Quantity Ordered to form the Extended Price (the total price for an individual order item). Note that this transformation does not lose information: the Unit Price for a particular order item can be derived by dividing the Extended Price by the Quantity Ordered.

Figure 13 shows the detailed fact table design for the transaction level (non-summarized) Order Item star schema. Each row in the table corresponds to an individual order item.

- The key of the fact table consists of the keys of all the dimension tables, including the degenerate dimension.
- The non-key attributes are the union of the attributes in the two original transaction entities (Order and Order Item). However, a non-additive fact (Unit Price) has been converted to an additive fact (Extended or Item Price = Order Quantity x Unit Price).

In this case, no data is lost from the original (normalized) data model—we can reconstruct details of individual orders down to the line item level.

For transaction entities connected in a "master-detail" structure, all attributes of the master record should be allocated down to the item level if possible (Kimball, 1995). For example, if discount is defined at the master (Order) level, the total discount amount should be allocated at the item level (e.g., in proportion to the price for each item). Otherwise, queries will result in multiple counting of discounts. The same should be done to delivery charges, order level taxes, fees, etc. If attributes of the master entity cannot be allocated down to the item level, separate star schemas may be required for each.

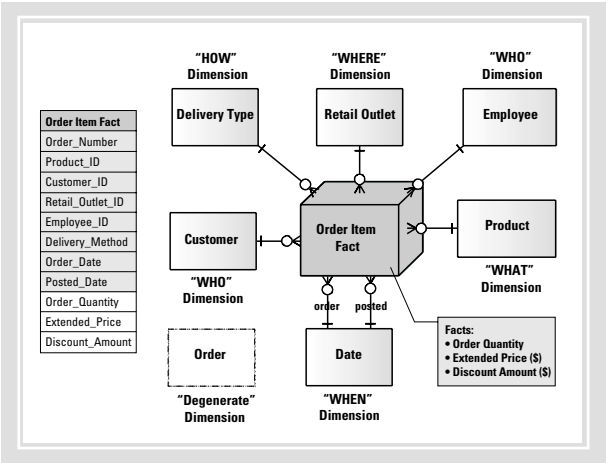

Figure 13. *Transaction Level Order Item Fact Table*

Figure 14 shows the detailed fact table design for the summary level star schema of Figure 12. Each row in this fact table summarizes sales by date, product, and retail outlet.

- The key of the fact table consists of the keys of all the dimension tables.
- The non-key attributes of the fact table are all summarized facts. Total Sales Quantity is the total daily quantity sold for each product in each retail outlet (calculated as the sum of Order Quantity grouped by Date, Product ID, and Retail Outlet ID). Total Sales Amount is the total daily revenue for each product in each retail outlet (calculated as the sum of Order Quantity x Unit Price). Total Discount Amount is the total daily discount by product and retail outlet (the sum of Discount Amount). An additional fact, Number of Orders, counts the number of separate orders by day, product, and retail outlet, which is a semi-additive fact. This can be added across Retail Outlets and Dates but not across Products, as this would result in double counting. Including such a fact is questionable, as it could lead to erroneous conclusions by end users.

Such summarization loses information compared to the original (normalized) model: we cannot reconstruct details of individual order items from this star schema. This means that some queries are not possible: for example, it is possible to determine the total sales of a product on any given day, but not the characteristics of the customers who bought them.

In some cases, fact tables may contain no numerical facts at all. For example, in a star schema that records the incidence of crimes, the fact table may record when and where a crime took place, the type of offense committed, and who committed it. However, there are no measures involved—the important information is simply that the crime took place. This is called a factless fact table. Other examples of factless fact tables include traffic accident, student enrollment, and disease statistics. The only relevant aggregation operation for such types of events is COUNT, which can be used to analyze the frequency of events over time.

Step 4: Detailed Dimension Table Design

In this step, we complete the detailed design for dimension tables in each star schema (attribute level design). This completes the detailed design of the star schema.

4.1 Define Dimensional Key

The key of each dimension table should be a simple (single attribute) numeric key. In most cases, this will just be the key of the underlying component entity. However, the operational key may need to be general-

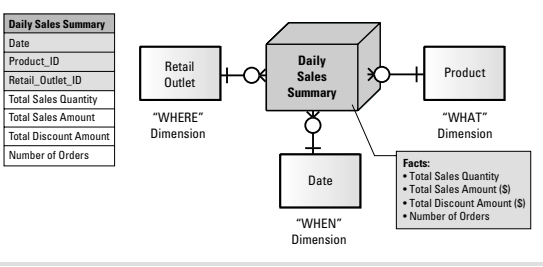

Figure 14. *Daily Sales Summary Fact Table*

ized to ensure that it remains unique over time. Because operational systems often only require key uniqueness at a point in time (i.e., keys may be reused over time), this may cause problems when performing historical analysis. Another situation where the dimensional key needs to be generalized is in the case of slowly changing dimensions.

4.2 Collapse Hierarchies

Dimension tables are formed by "collapsing" or denormalizing hierarchies (defined by classification entities) into component entities. Figure 15 shows how the hierarchies associated with the Customer component are collapsed to form the Customer dimension table. The resulting dimension table consists of the union of all attributes in the original entities—as many descriptive attributes should be included as possible to support slicing and dicing of data in different ways. In practice, it is common for dimension tables to contain more than a hundred attributes. This process introduces redundancy in the form of transitive dependencies, which are violations to third normal form (3NF) (Codd, 1970). This means that the resulting dimension table is in second normal form (2NF).

4.3 Replace Codes and Abbreviations by Descriptive Text

To make the star schema as understandable as possible, codes and abbreviations in source data should be removed and replaced by descriptive text (Kimball, 1996). While such codes and abbreviations have an important place in efficient processing of transactions at the operational level, they simply obfuscate things in the data warehousing environment. Use of such codes and abbreviations places a cognitive load on the end user to remember what they mean, something users can well do without. Codes and abbreviations may be optionally retained in the star schema if they add to understanding.

Figure 16 shows the detailed star schema design for the Order transaction (line item granularity). This shows attributes and keys for the fact table and all dimension tables:

- All of the tables in the star schema are denormalized to at least some extent: each table corresponds to multiple entities in the original normalized ER model. The dimension tables are the result of collapsing classification entities into component entities, while the fact table is the result of merging the master and detail transaction entities.
- All of the dimension tables are in 2NF: they have simple keys and no repeating attributes. The fact table is in 3NF as Discount Amount (an attribute of the Order entity) has been allocated down to the line item level. This means that Order becomes a degenerate dimension as it has no other dependent attributes.
- The Date Dimension is a new table which does not correspond to any entity in the original ER model. This is because dates must be explicitly modeled in a star schema, whereas at the operational level they are represented as data types.

Summary

This article has examined the nature of dimensional modeling and proposed a possible theoretical

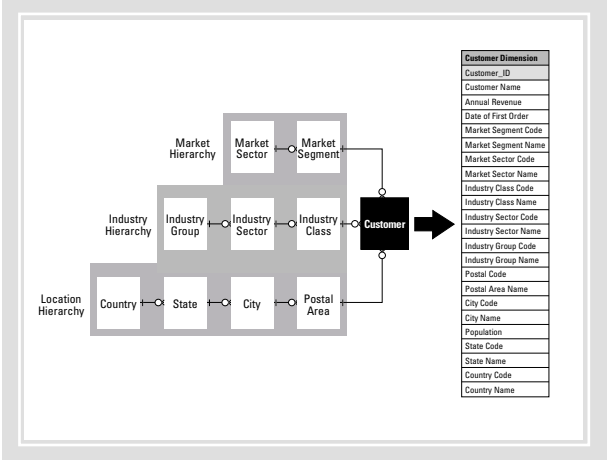

Figure 15. *Collapsing Hierarchies*

explanation for why it has been so successful in practice. It has also described an approach for deriving dimensional models directly from ER models. This provides a "bridge" between operational system (OLTP) design and data warehouse (OLAP) design, which makes it easier for people trained in traditional database design techniques to learn dimensional design.

Deriving dimensional models from an ER model provides a more structured approach to developing dimensional models than starting from first principles, which can help avoid many of the pitfalls faced by inexperienced designers. It also helps to resolve the difficult problem of matching "supply" (operational data sources) and "demand" (end-user information needs) in data warehouse design. Finally, it results in a more complete dimensional design, as it involves selecting from all possible ways of analyzing the data in the underlying operational systems, and is therefore less dependent on the designer's ability to choose the "right" dimensions.

Relationship between ER Modeling and Dimensional Modeling

In this paper, we have challenged the widely accepted view that dimensional modeling is fundamentally different from, and incompatible with, ER modeling. We have shown that a dimensional model is just a restricted form of an ER model and there is a straightforward transformation between the two. An ER model can be transformed into a set of dimensional models by a process of selective subsetting, denormalization, and (optional) summarization:

- Subsetting: The ER model is partitioned into a set of separate star schemas, each centered on a single business event (transaction entity). This reduces complexity through a process of "chunking."
- Denormalization: Hierarchies in the ER model are collapsed to form dimension tables. This further reduces complexity by reducing the number of separate tables. Master-detail transaction entities are denormalized into a single fact table, as these represent different views of the same underlying event.
- Summarization: The most flexible dimensional structure is one in which each fact represents a single transaction or line item. However, summarization may be required for performance reasons, or to suit the needs of a particular group of users.

At the detailed (attribute) design level, a range of transformations are required:

- Generalization of operational keys to ensure uniqueness of keys over time and/or to support slowly changing dimensions.
- Conversion of non-additive or semi-additive facts to additive facts.
- Substitution of codes and abbreviations with descriptive text.

Advanced Design Issues

The transformation approach described in this article results in a fully detailed star schema design. However, in practice, dimensional modeling tends to be an iterative process. The first-cut dimensional model will almost certainly need refinement—as in most design problems, there are always exceptions and special cases to deal with. In Part II of this discussion, (publishing in the Fall issue of the *Business Intelligence Journal*), we will discuss advanced design issues that need to be considered in dimensional modeling:

- Alternative dimensional structures: stars, snowflakes, and starflakes
- Slowly changing dimensions
- Minidimensions
- Heterogeneous star schemas
- Dealing with non–hierarchical data

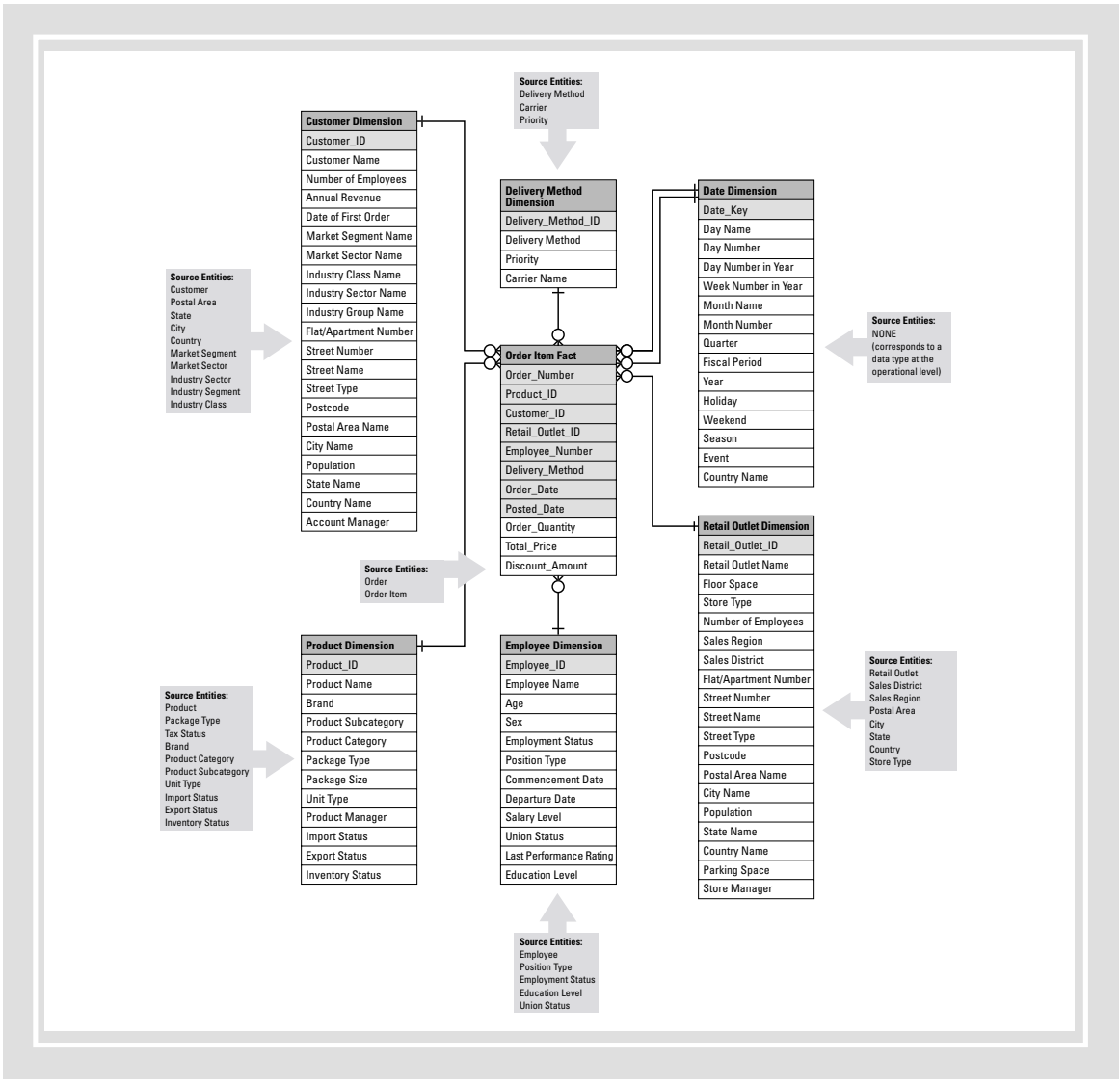

Figure 16. *Order Item Star Schema (fully attributed)*

REFERENCES

Baddeley, A. D. "The Magical Number Seven: Still Magic After All These Years?" **Psychological Review**, 101, 1994.

Baddeley, A. D., **Essentials of Human Memory**, Psychology Press, Hove [England], 1999.

Chen, P. P. "The Entity Relationship Model: Towards An Integrated View Of Data," **ACM Transactions On Database Systems**, 1, 9-36, 1976.

Codd, E. F. "A Relational Database Model For Large Shared Data Banks," **Communications of the ACM**, 13, 377-387, 1970.

Craig, R. "The Schema Wars," **ENT**, 3, 30-32, 1998.

Flood, R. L. and Carson, E. R. **Dealing With Complexity: An Introduction To The Theory And Application Of Systems Science**, Plenum Press, 1993.

Gallas, S. "Put Data in its Place!" **DM Direct Business Intelligence Newsletter**, 1999.

Glassey, K. "Seducing the End User," **Communications of the ACM**, 1998.

Kent, W. "A Simple Guide To The Five Normal Forms Of Relational Database Theory," **Communications Of The ACM**, 1983.

Kimball, R. "Is ER Modeling Hazardous to DSS?" **DBMS Magazine**, 1995.

Kimball, R. **The Data Warehouse Toolkit**, John Wiley and Sons, New York, 1996.

Kimball, R. "A Dimensional Manifesto," **DBMS Online**, 1997.

Kimball, R. **The Data Warehouse Toolkit: The Complete Guide to Dimensional Modeling**, John Wiley and Sons, New York, 2002.

Klir, G. J. **Architecture of Systems Problem Solving**, Plenum Press, New York, 1985.

Kuhn, T. S. **The Structure Of Scientific Revolutions**, University Of Chicago Press, 1970.

Lipowski, Z. J. "Sensory And Information Inputs Overload: Behavioural Effects," **Comprehensive Psychiatry**, 16, 1975.

Maier, R. (1996) "Benefits And Quality Of Data Modeling-Results Of An Empirical Analysis," **In Proceedings Of The Fifteenth International Conference On The Entity Relationship Approach** (Ed, Thalheim, B.), Elsevier, Cottbus, Germany.

Miller, G. A. "The Magical Number Seven, Plus Or Minus Two: Some Limits On Our Capacity For Processing Information," **The Psychological Review**, 1956.

Moody, D. L. (2002) "Complexity Effects On End User Understanding Of Data Models: An Experimental Comparison Of Large Data Model Representation Methods," **In Proceedings of the Tenth European Conference on Information Systems** (ECIS, 2002) Gdansk, Poland.

Newell, A. A. and Simon, H. A. **Human Problem Solving**, Prentice-Hall, 1972.

Oates, J., "The Great Data Warehouse Debate," **Sybase White Paper**, 2000.

Raisinghani, M. S., "Adapting Data Modeling Techniques for Data Warehouse Design," **Journal of Computer Information Systems**, 40, 2000.

Simon, H. A. **Sciences Of The Artificial** (3rd edition), MIT Press, 1996.

Spencer, T. and Loukas, T. "From Star to Snowflake to ERD," **Enterprise Systems Journal**, 1999.

Uhr, L., Vossier, C. and Weman, J. "Pattern Recognition over Distortions by Human Subjects and a Computer Model of Human Form Perception," **Journal of Experimental Psychology: Learning**, 63, 1962.

Weber, R. A. "Are Attributes Entities? A Study Of Database Designers' Memory Structures," **Information Systems Research**, 1996.

Winter, R. and Strauch, B. "Demand-Driven Information Requirements Analysis in Data Warehousing," **Journal of Data Warehousing**, V8N1, 2003.