Dimensional Data Modeling

This document is the creation of Sybase and/or their partners.

Thomas J. Kelly
Principal Consultant
Data Warehouse Practice
Sybase Professional Services

Increasingly, competition and leaner profits are driving organizations to find more efficient methods for finding customers, meeting their needs, and keeping them. If your business was the old-time neighborhood market, you'd know your customers from your personal interactions with them. However, if your business provides services to millions of people every year, it's impossible to get personal with each one.

We can, however, look at the data around us to learn more about our customers and determine what motivates them to use our products and services. Organizations are looking to data warehouses to provide those answers. But it starts with being able to ask the right questions. This paper describes one set of techniques for organizing warehouse data to facilitates users' ability to ask, and get answers to, the right questions.

Warehouse Design Using Dimensional Data Modeling

If you've ever watched a builder constructing a house, you'll notice that he has a belt with several tools on it. Each tool has a specific purpose - each contributing to the completing the total job. You'd probably notice that he has more than one hammer. He probably also has a nail gun. Why so many tools, when his major job is to put a nail through wood?

A nail gun will do a great job on putting up wall studs, but is a little too indelicate for door moulding. A finishing hammer can handle the moulding, but won't do much for driving 12" spikes into the braces of a retaining wall. The builder picks the tool that best suits the job.

A designer chooses from among design techniques to select the best that suits the job. One technique that is gaining popularity in data warehousing is **dimensional data modeling**. For some data analysis situations, it can meet an organization's requirements for organizing warehouse data.

In OLTP systems, transactions tend to be made up of small, pre-defined queries. The programmer or the DBA develops, tests, and optimizes each query. The end-user rarely has to write queries or interact directly with the physical implementation of the data model.

In Data Warehousing environments, queries tend to use more tables, return larger result sets and run longer. Since these queries are typically unplanned and may not be reused, there is little or no opportunity for comprehensive optimization activities.

Non-technical users are often asked to comprehend how to relate four, six, ten, or more tables to answer a question. Further, performance suffers when many large tables make up the query and indexes have not been defined to optimize the query's access options.

The "Cube" metaphor provides a new approach to visualizing how data is organized. A cube gives the impression of multiple dimensions. Cubes can be 2, 3, 4, or more dimensions. Users can "slice and dice" the dimensions, choosing which dimension(s) to use from query to query.

Each dimension represents an identifying attribute. The cube shape indicates tha.

- several dimensions can be used simultaneously to categorize facts
- the more dimensions used, the greater the level of detail retrieved
- dimensions can also be used to constrain the data returned in a query, by restricting the rows returned to those that match a specific value or range of values for the constraining dimension

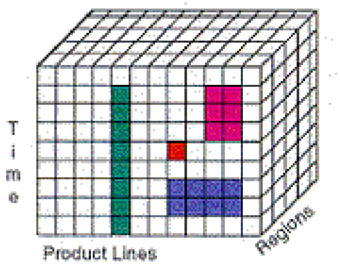


Figure 1: New Approach to Visualizing How Data is Oraganized

A warehouse user may be interested in seeing the Total Sales by Sales Region. Responding to the query, we might see the aggregated Sales (the facts) associated with the Region dimension as shown below. A single dimension (Region) is used to obtain an aggregation of a related fact (Sales).

Region	**Sales**
East	$1,552M
West	$1,732M

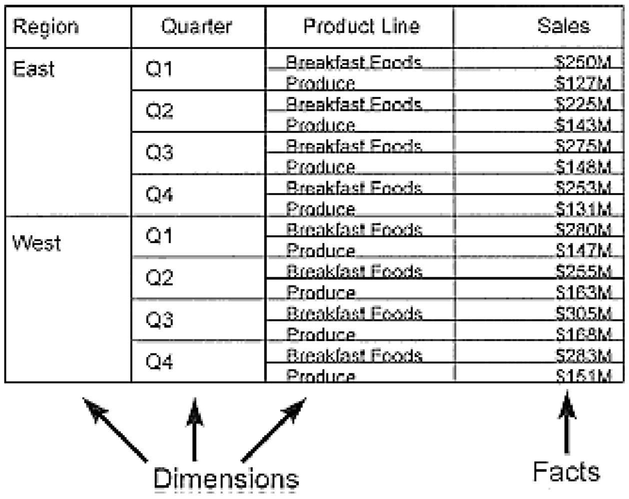
The warehouse user may want to "drill-down" and see the Total Sales for each Region by Sales Quarter. (For ease of understanding, the query tool has replaced the "Time" dimension title with the word "Quarter"). Here we see the aggregated facts associated with each combination of the Region and Time dimensions.

Region	**Quarter**	**Sales**
East	Q1	$377M
	Q2	$368M
	Q3	$423M
	Q4	$384M
West	Q1	$427M
	Q2	$418M
	Q3	$453M
	Q4	$434M

Finally, we see that if we use three dimensions, the aggregated facts are associated with each combination of the Region, Time, and Product Line dimensions. Each additional dimension increases the level of detail.

Region	Quarter	Product Line	Sales
East	Q1	Breakfast Foods	$250M
		Produce	$127M
	Q2	Breakfast Foods	$225M
		Produce	$140M
	Q3	Breakfast Foods	$275M
		Produce	$148M
	Q4	Breakfast Foods	$253M
		Produce	$131M
West	Q1	Breakfast Foods	$280M
		Produce	$147M
	Q2	Breakfast Foods	$255M
		Produce	$163M
	Q3	Breakfast Foods	$305M
		Produce	$148M
	Q4	Breakfast Foods	$283M
		Produce	$151M

This "flattened" view of the dimensional data unfolds the cube by each dimension and the facts are aggregated at the intersection of the chosen dimensions.



Region	Quarter	Product Line	Sales
East	Q1	Breakfast Foods	$250M
		Produce	$127M
	Q2	Breakfast Foods	$225M
		Produce	$143M
	Q3	Breakfast Foods	$275M
		Produce	$148M
	Q4	Breakfast Foods	$253M
		Produce	$131M
West	Q1	Breakfast Foods	$280M
		Produce	$147M
	Q2	Breakfast Foods	$255M
		Produce	$163M
	Q3	Breakfast Foods	$305M
		Produce	$168M
	Q4	Breakfast Foods	$283M
		Produce	$151M

Dimensions
Facts

Figure 2 The Flattened View of the Dimensional Data Cube

The Star Schema

Dimensional data modeling adds a set of new schemas to our logical modeling toolkit. The first is called a **star schema**, named for the star-like arrangement of the entities.

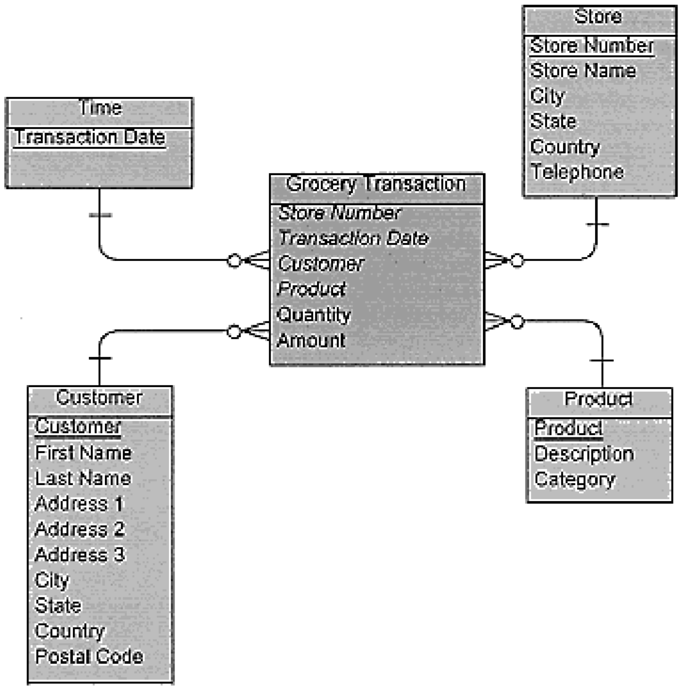


Figure 3 The Star Schema

A Star Schema uses many of the same components that you would find in any Entity-Relationship Diagram:

- Entities
- Attributes
- Relationship Connections
- Cardinality
- Optionality
- Primary Keys

The center of a star schema is the Fact Table. It is the focus of dimensional queries -- it's where the real data (the facts) are stored. Facts are numerical attributes, such as counts and amounts, that can be summed, averaged, max'ed, min'ed, and be aggregated by a variety of statistical operations. Fact attributes contain measurable numeric values about the subject matter that is managed by the Fact table. A Grocery Transaction could be a line on a register tape, such as three bags of flour, totaling $11.70.

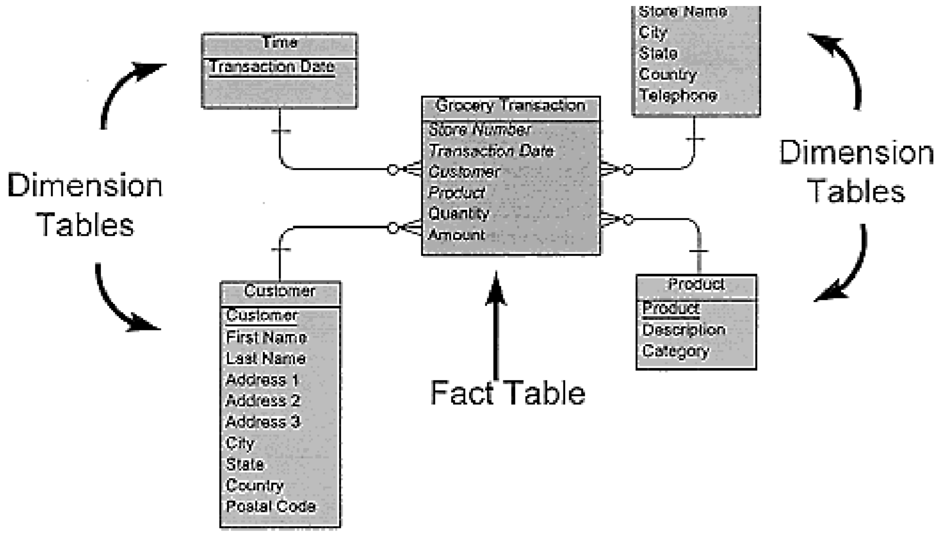


Figure 4 The Anatomy of the Star Schema

Dimensional attributes provide descriptive information about each row in the Fact table. These attributes are used to provide links between the Fact table and the associated Dimension tables. Over the next few paragraphs, you'll see how the Dimension tables in this star schema relate to descriptive attributes in the Fact table.

Time is a critical component of the data warehouse data model. Data in an OLTP system tends to be focused on current values. DSS environments allow us to analyze data and how it changes over time:

- Sales this period over last period
- Purchasing trends

We'll cover more details about modeling the time dimension later in this paper.

The *Store* dimension allows us to categorize transactions by store, including location of the store and its relation to other geographically-distributed stores. The *Product* dimension allows us to analyze purchasing patterns by product and groupings of products. The *Customer* dimension allows us to analyze purchases by customer, such as purchasing frequency (including which percentages of customers purchase at which frequencies) and purchases by customer locations (indicating how many customers are willing to travel some distance to shop at a particular store).

Dimension tables are used to guide the selection of rows from the Fact table.

A star schema works properly when a consistent set of facts can be grouped together in a common structure, called the Fact table, and descriptive attributes about the facts can be grouped into one or more common structures, called Dimension tables.

Consider the logical model for a retail store in Figure 4. Queries that can be answered by this model would include:

> *'How many 25oz boxes of Trix were sold in March by Boston stores to customers living in Natick?'*

The intersection of each of these dimensions will determine which rows will be aggregated to answer the query.

Fact tables typically have large volumes of rows, while dimension tables tend to have a smaller numbers of rows. The key advantage to this approach is that table join performance is improved when one large table can be joined with a few small tables. Often the dimension tables are small enough to be fully cached in memory.

When you compare the star schema to a fully normalized relational design, you can see some significant differences:

- The star schema makes heavy use of denormalization to optimize for speed, at a potential cost of storage space. The normalized relational design minimizes data duplication, reducing the work that the system must perform when data changes.
- The star schema restricts numerical measures to the fact table(s) while the normalized

relational design can store transactional and reference data in any table.

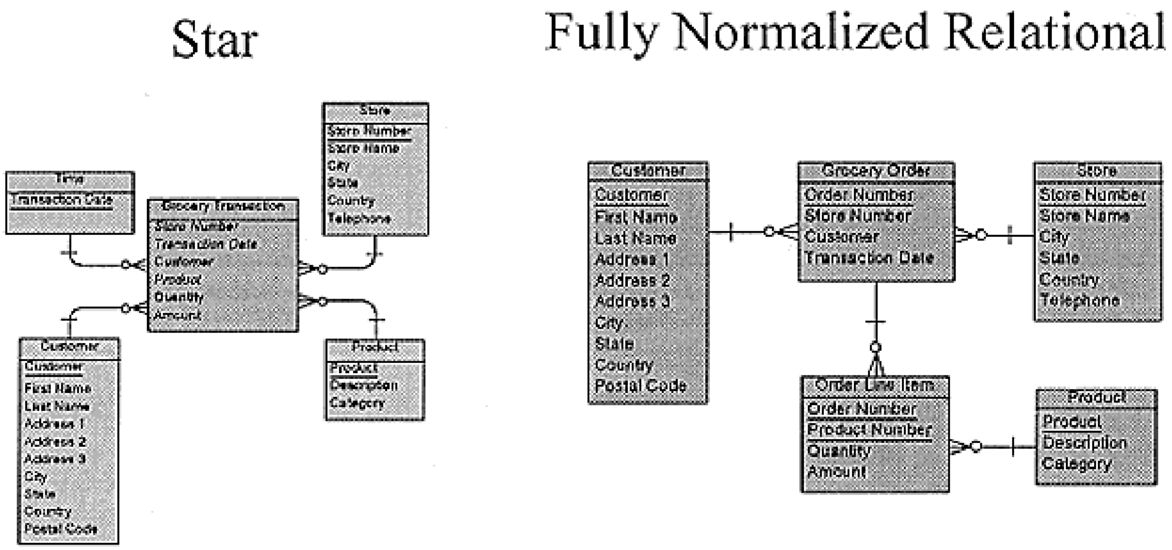


Figure 5 Comparing Design Schemas

The Snowflake Schema

A Snowflake schema adds hierarchical structures to the dimension tables, such as a Region dimension table that is connected to a Store table. Or a Product Category table connected to a Product table.

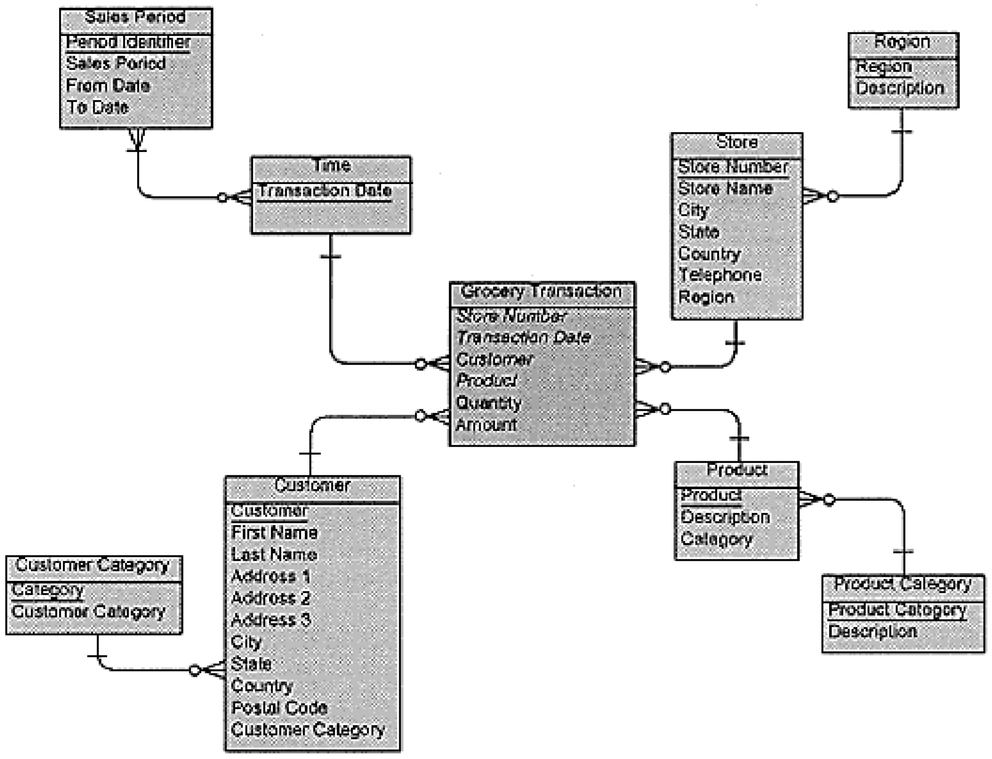


Figure 6 Comparing Design Schemas

Now the schema recognizes that some number of stores make up a particular sales region. Several different products make up a Product Category. Queries that can be answered by this model would include:

'What were the total sales of breakfast cereal to suburban, no-kids adults at east coast stores in Q4 1995?'

You can see that this schema can be very powerful in processing abstractions (less-specific layers of the hierarchy). Customer categories could be based on particular purchasing patterns, credit history, or demographic categories.

Don't get the impression that you can't do this analysis in a normalized relational model, because you certainly can. Many designers have developed relational data models for DSS that included these types of data. Dimensional modeling merely puts more emphasis on supporting this type of analysis.

Multiple 'parent' dimensions may be defined in the snowflake schema. This data model adds another time-based dimension, Promotion Period, and a product-related dimension, Product Line.

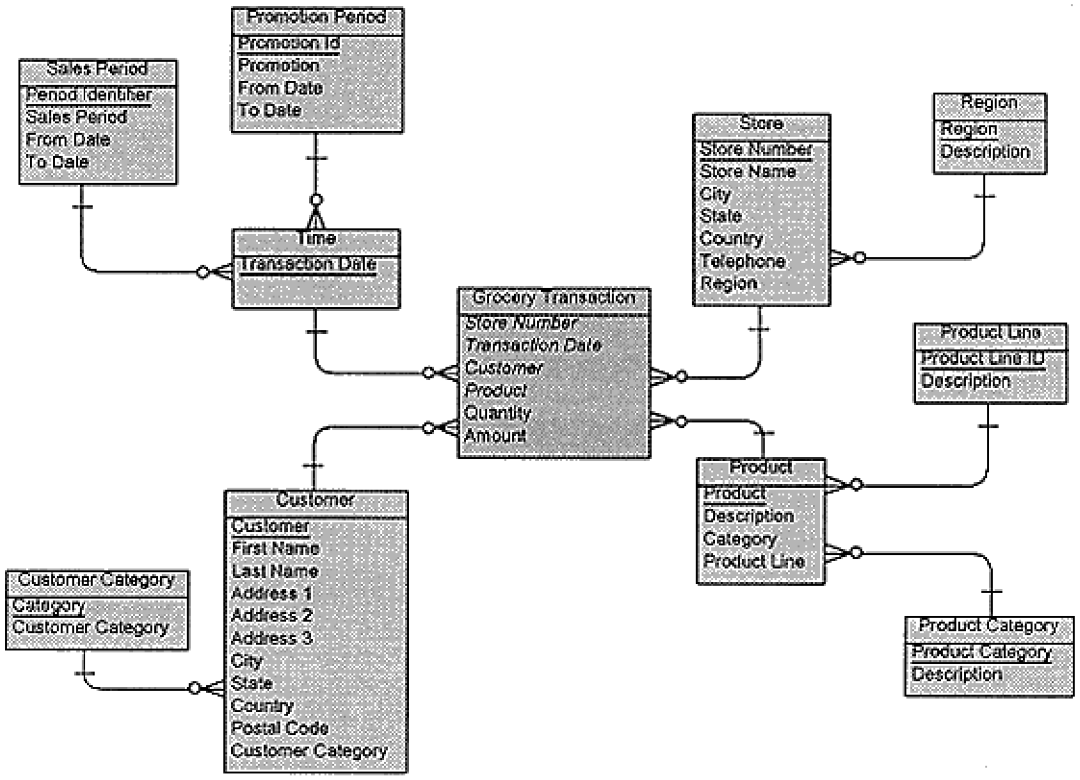


Figure 7 Multiple Parent Dimensions in a Snowflake Schema

Product Category may refer to Frozen Foods, while Product Line may refer to Diet Foods. Both are characteristics that may or may not be related. For example, there are certainly Frozen Diet Foods, but there are also Canned Diet Foods as well as Frozen Non-Diet Desserts.

Examining the time-related parent dimensions, we want to be able to define the periods during which special price reductions were offered. A separate dimension table is defined to describe these promotion periods. This new table allows us to have promotion periods that can span selling periods. You can even support overlapping promotions.

In our dimensional data model, not all stores will participate in all promotions at the same time. Further, not all products will be included in a particular promotion.

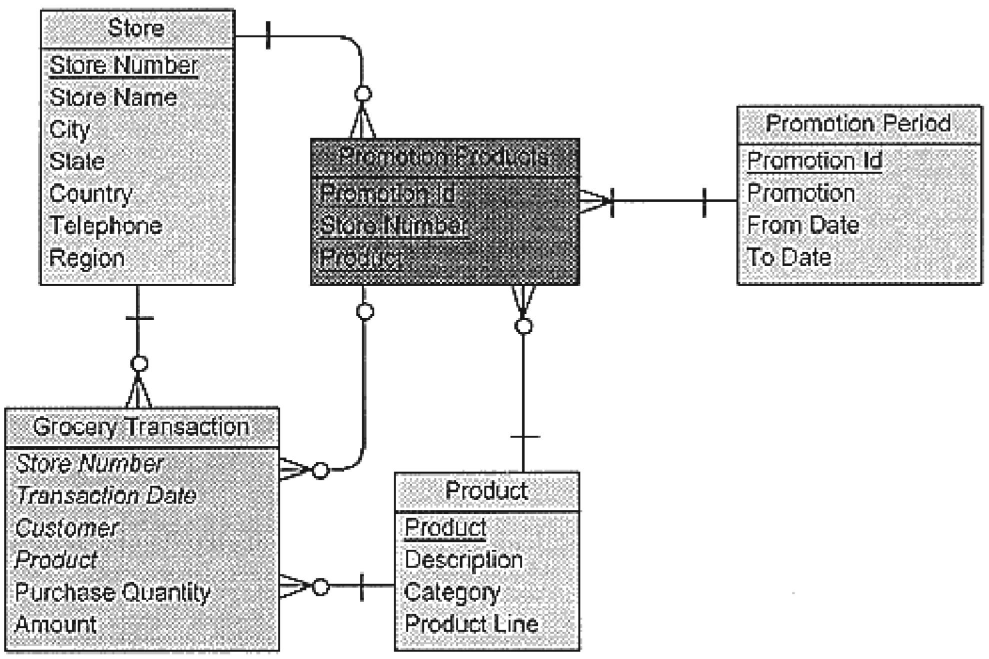


Figure 8: Existence Tables

In this case, an **Existence Table** is used to define which products will be offered at which stores during which promotions. This table will assist the analyst in evaluating the effectiveness of promotions by identifying the participating stores and products. Similar to associative tables found in a normalized relational model, existence tables manage exceptions - where certain instances of one table are related to certain instances of one or more other tables.

An Extended Snowflake Schema makes use of multiple fact tables. As data models become more comprehensive, more fact tables will be defined. Further, large queries may make use of two or more fact tables, linked by multiple dimension tables.

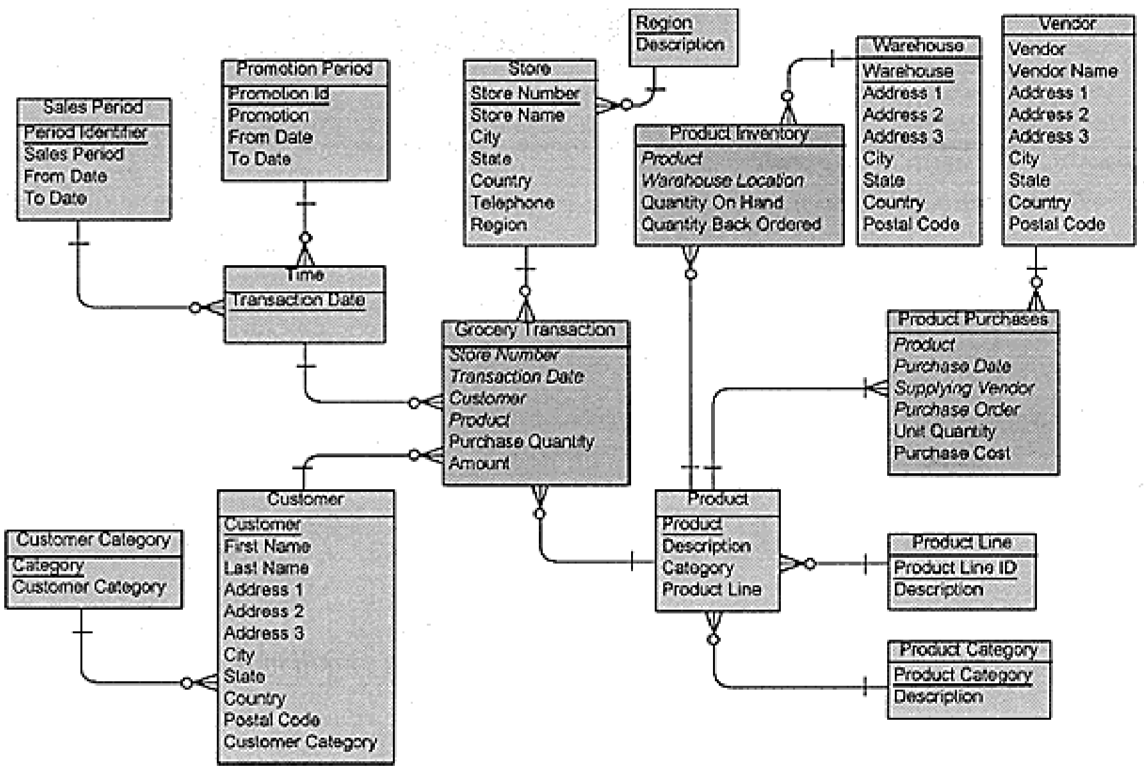


Figure 9: Extended Snowflake Schema

Modeling the Time Dimension

Some organizations will use a variety of time periods for data analysis. Some of these will be 'artificial' time periods, such as fiscal periods vs. calendar periods.

In some countries, work weeks start on a Monday and end on a Sunday, while other countries start counting the week on Sunday. On the other hand, calendar months and years sometimes start and end on any day of the week.

The approach used for modeling the time dimension will strongly depend on how the organization will analyze the warehouse data:

- Does the organization have fiscal and calendar years?
- Can different stores have different fiscal periods?
- Do all stores have the same promotions at the same times?
- Can different stores have similar promotions at different times?
- Do some communities recognize holidays that others do not?
- Are all stores open every day of the year?
- Are all stores open the same hours of the day?
- Will every store be open the same hours every day?

A traditional relational model will tend to fully normalize the time dimension. This model minimizes data duplication. All table rows are very narrow, but the larger number of tables may result in more tables to join.

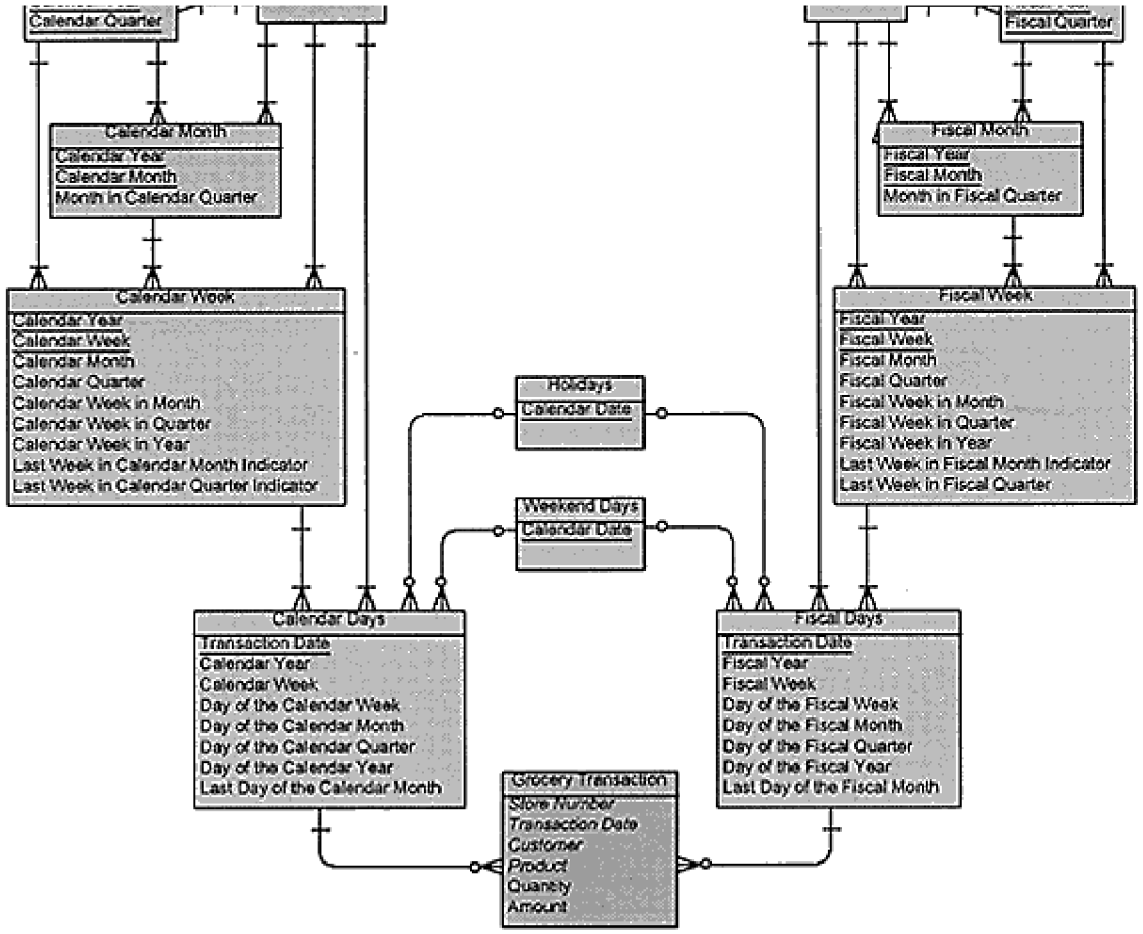


Figure 10: Modeling Time in a Normalized Relational Model

By completely denormalizing the time dimension, the star schema is able to significantly reduce the number of tables joined in a time-based query.

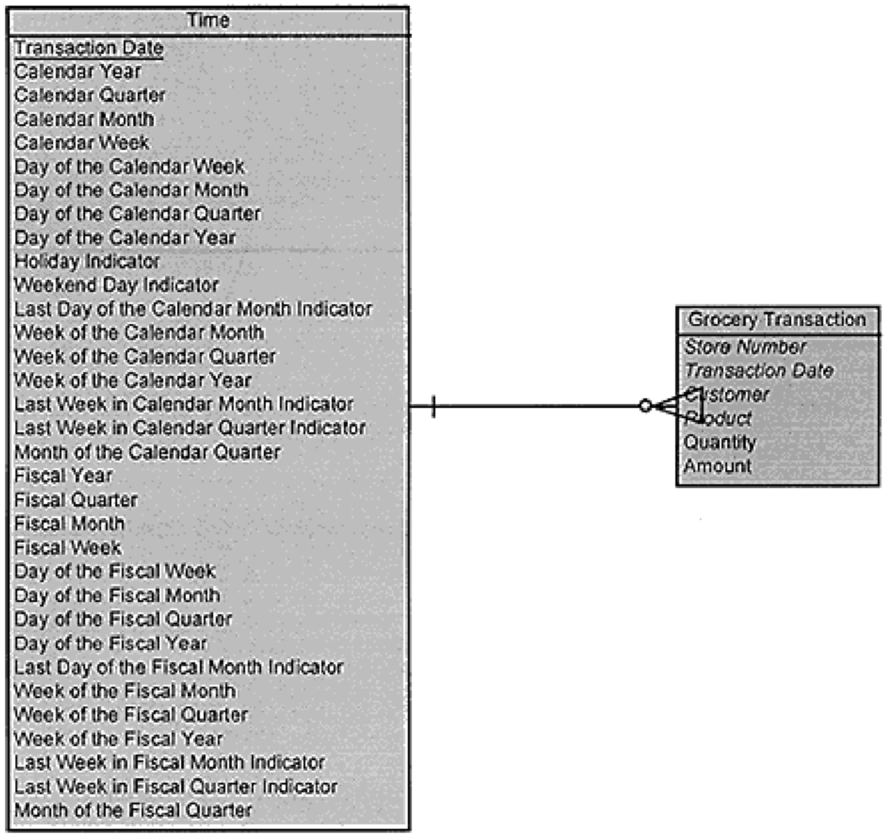

Time
Transaction Date
Calendar Year
Calendar Quarter
Calendar Month
Calendar Week
Day of the Calendar Week
Day of the Calendar Month
Day of the Calendar Quarter
Day of the Calendar Year
Holiday Indicator
Weekend Day Indicator
Last Day of the Calendar Month Indicator
Week of the Calendar Month
Week of the Calendar Quarter
Week of the Calendar Year
Last Week in Calendar Month Indicator
Last Week in Calendar Quarter Indicator
Month of the Calendar Quarter
Fiscal Year
Fiscal Quarter
Fiscal Month
Fiscal Week
Day of the Fiscal Week
Day of the Fiscal Month
Day of the Fiscal Quarter
Day of the Fiscal Year
Last Day of the Fiscal Month Indicator
Week of the Fiscal Month
Week of the Fiscal Quarter
Week of the Fiscal Year
Last Week in Fiscal Month Indicator
Last Week in Fiscal Quarter Indicator
Month of the Fiscal Quarter
Grocery Transaction
Store Number
Transaction Date
Customer
Product
Quantity
Amount

Figure 11: Modeling Time in a Dimensional Model

There are fewer tables to join when using time indicators, but you can end up with very wide rows to accommodate all of the possible indicators needed (calendar and fiscal calendars). Further, some flexibility is reduced:

- The Time row must be expanded further to handle other types of time periods
- Several relationships (Month in Quarter) are duplicated frequently

Aggregates

This paper has already discussed managing data at the lowest-level of detail available, such as individual transactions. Aggregates are statistical summaries of these details. You can define one or more levels of aggregates of the same facts.

Aggregates give the perception of improved query performance. You can return query results faster if you use 10 pre-calculated aggregate rows to answer the query, rather than 10,000 low-level detail rows.

A grocery store chain might want to aggregate daily sales by store, metro area, and region. The chain might also want to aggregate weekly sales by categories, such as frozen foods and produce, by metro area. Trend analysis at the transaction level might be too detailed and take too long to return query results. Maintaining these aggregates could make significant performance improvements in analysis activities.

Using the retail grocery dimensional data model, we can add an **aggregate fact table**. This table contains pre-calculated subtotals of the detail data found in the Grocery Transaction table.

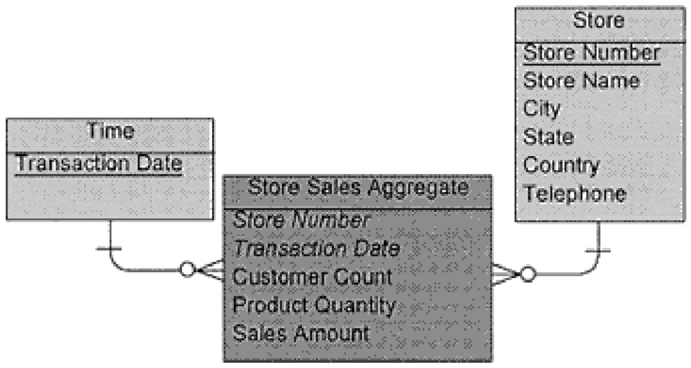


Figure 12: A Two-Dimension Aggregate Table

During an aggregation process, the aggregate table is populated with the sums of the day's facts for each store:

- Customer Count - How many customers purchased products?
- Product Quantity - How many product units were sold?
- Sales Amount - What was the total product sales for the day?

In addition, other statistical facts can be derived from these aggregates:

- Average sales per customer
- Average unit quantity per customer

We can add another aggregate table that accumulates the day's sales facts along three dimensions: Metro Area, Product Category, and Time.

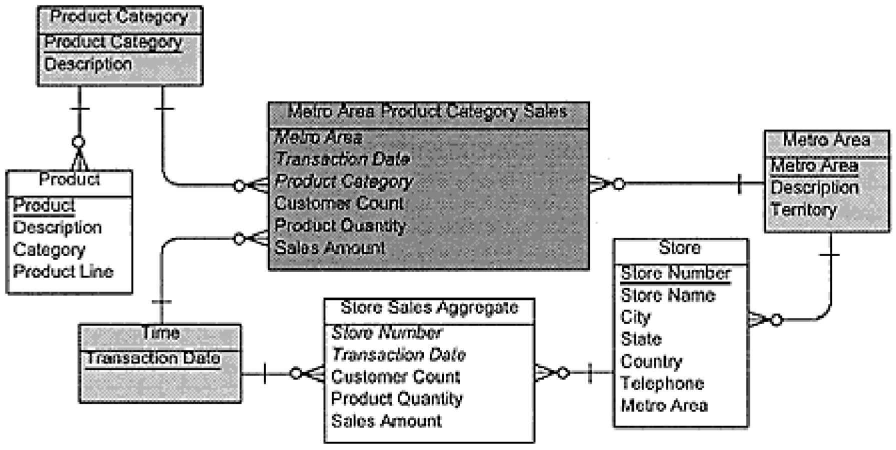


Figure 13: A Three-Dimension Aggregate Table

This table contains aggregates of several facts by Product Category by Metro Area for each Day:

- Customer Count - How many customers purchased products?

- Product Quantity - How many product units were sold?
- Sales Amount - What was the total product sales for the day?

We can add any number of aggregate tables to our data model. However, to avoid unnecessary proliferation of aggregates, we need some rules to guide our definition of aggregates. We'll discuss the need for guidelines later in this paper.

Issues in Dimensional Data Modeling

This section began by describing that tools tend to be specialized. The right tool for the right job can be very powerful. The wrong tool can cause some real headaches. There are several challenges to face when using dimensional data modeling techniques:

- **Denormalization**
- **Queries Using Multiple Fact Tables**
- **Dimension Table Data Volumes**
- **Managing Aggregates**
- **Snowflakiness**
- **Data Sharing**

Denormalization:Dimensional data modeling requires significant denormalization of data in the data warehouse to avoid table joins. But denormalization issues are already well understood:

- Denormalization causes data duplication, emphasizing performance over redundancy in space allocations
- More work is done when duplicate data changes, such as defining time relationships in a new fiscal year or when a business unit is shifted to a new region
- A denormalized data model is appropriate for OLTP, where uses of the data is predictable and can be planned, but does not provide the flexibility for addressing unplanned requirements

Denormalization is a technique that is often used for improving query performance but it requires advanced knowledge of how the data will be used. In some situations, denormalization techniques employed to achieve performance expectations required a five-fold increase in storage requirements because of the magnitude of data duplication. Other situations have increased storage requirements by an order of magnitude or more.

Denormalization tends to reduce the flexibility of the data warehouse, and therefore, limits the kinds of queries that can be executed against the data warehouse. For example, a fully normalized data model separates each entity and limits the amount of information about each entity's relationship to other entities. A denormalized data model combines several entities, requiring that the denormalized entity include assumptions about the relationships of the major categories of the new entity.

In a fully normalized data model, a store resides within a community. The Store entity does not contain any other intelligence about higher levels in the location hierarchy. In a denormalized data model, the entity must contain all relation ship information. A change to this hierarchy cannot be isolated to a particular part of the model, it affects the entire hierarchy entity.

When duplicated data changes, there may be dozens, hundreds, or thousands of rows that must be corrected in a denormalized schema. Often, a normalized relational schema will restrict these changes to a single row in a single table.

Queries Using Multiple Fact Tables: As a dimensional data model grows to include multiple fact tables, there is a tendency to want to use more than one fact table in a query (such as the simple model shown in Figure 9).

For example, you might want to compare Product Purchases to Grocery Transactions for certain products on certain days. That might tell you if your purchases are keeping pace with your sales, and vice versa.

To process this query, the dimensional data model must include two or more snowflakes. This "snow storm" schema uses a technique called "drill across" to combine data from separate fact tables. Since fact tables tend to have a large number of rows, this technique defeats one of the major strengths of the dimensional data model: improved performance by restricting joins to one large fact table with one or more small dimension tables.

Dimension Table Data Volumes: When joining database tables in a star schema, the dimension tables are joined first. The intersection of these tables are then processed against the fact table. If the dimension tables are small, or queries are very selective in the number of qualifying rows in the dimension table, a similarly small intermediate result set will be processed against the fact table.

On the other hand, one or more large dimension tables could produce a huge intermediate result set, significantly increasing query response time. Performance ends up being worse than traditional RDBMS table joins, further worsened by the wider rows that resulted from denormalization.

Managing Aggregates: Aggregates are pre-calculated answers to known queries. You cannot support unplanned, ad hoc queries with aggregates. (If you already knew that you needed certain aggregates, the queries that could use them would have been at least partially planned already). So unplanned queries may require orders of magnitude more time to answer.

By creating too many aggregate tables, you can encounter **Aggregate Table Explosion**. In a data model with a single fact table and 10 dimensional attributes, there are over 3 million possible aggregate tables that you can build. (10! = 3.63 million). Few organizations have the processing capacity to manage all of the potential aggregates from even this small model.

Similarly, using too many dimensions can cause **Row Count Explosion**. Consider a retail grocery chain that has 100 stores, 25 product categories, 50 customer types, and one year of daily sales data. As you can see from the table below, just four aggregates can translate into millions of rows of subtotals. In some cases, aggregates can make up over three quarters of the total warehouse space.

Aggregate	**Table Rows**
Monthly Store Sales	1,200
Daily Store Sales	36,500
Daily Store Sales by Category	912,500
Daily Store Sales by Category and Customer Type	45,625,000

When defining aggregates, consider using the 80:20 rule -- 80% of the performance improvement will be achieved by only 20% of the candidate aggregates. Further, these aggregates will be reused often enough that you will have achieved a reduction in the total computing resources used to process queries.

The remaining 80% of the candidate aggregates will fall into one of two categories:

- No reduction in computing resources to obtain query results -- you're merely scheduling part of the query processing to run during non-peak periods
- An increase in computing resources to maintain aggregates that are seldom used -- you have those aggregates there "just in case"

Snowflakiness: One of the implications of using hierarchical structures is that they can grow dramatically. In Figure 17-14, there are many geographical layers that can be used for grouping stores. Using the snowflake technique, dimension tables are added for each layer.

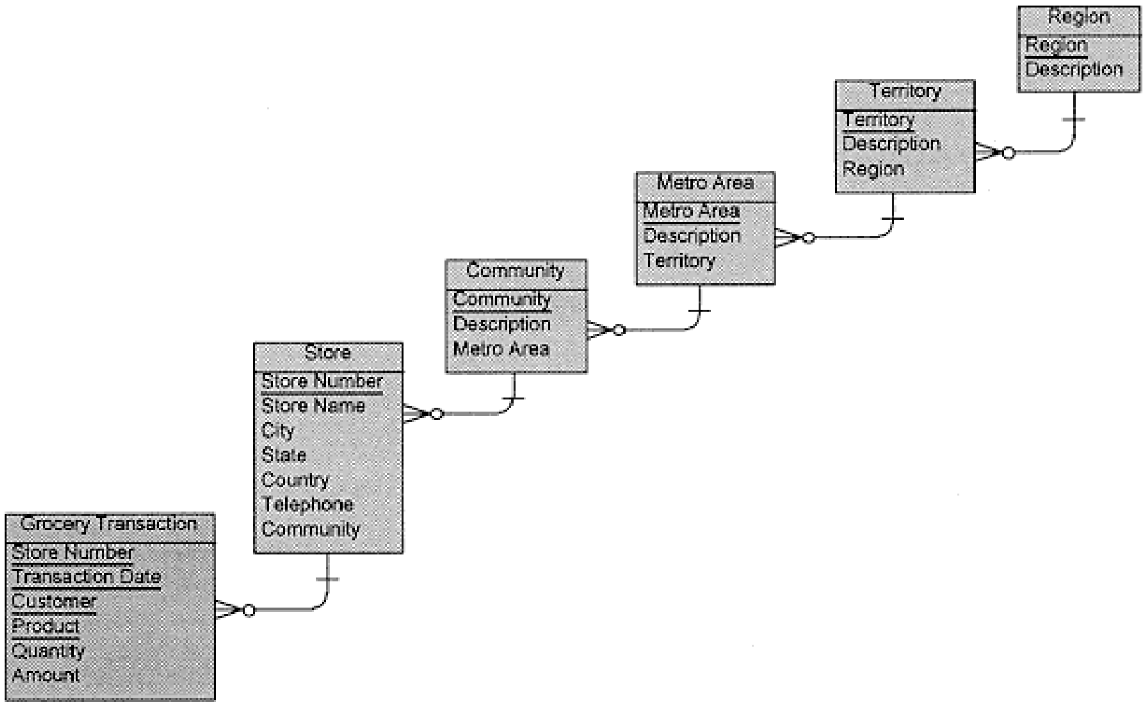


Figure 14: Increasing 'Snowflakiness'

However, all of these dimensions imply additional table joins, which can worsen query performance. Since the star schema emphasizes denormalization, the geographic dimensional hierarchy should be denormalized into the Store dimension table seen in Figure 15, eliminating all of the table joins possible on the previous model.

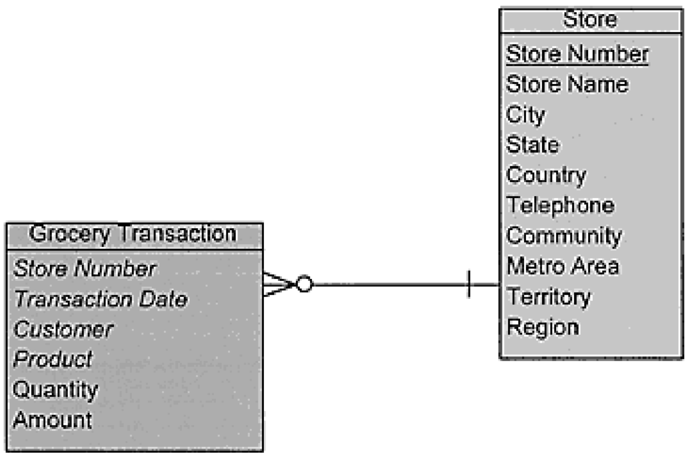


Figure 15: Denormalized Hierachy

The issue with this model is the duplication of data -- every relationship between Region and Territory is exponentially duplicated for every Metro Area, Community, and Store.

Data Sharing: To get optimal performance, some implementations of the dimensional data model will put each star schema into a separate database. Since some dimension tables will be used by more than one star, those tables must be duplicated in each database. This technique challenges the organization to provide additional storage for the extra copies of the data, as well as keeping the duplicate tables in sync.

In summary, dimensional data modeling is one of several techniques to keep in your data modeling toolkit. Like any tool, the dimensional data model is specialized to solve certain types of problems, though weak to solve others. When using the dimensional data model, simple decision support implementations are most likely to achieve high-performance goals.