P.G. Department of Computer Sciences,
University of Kashmir

Data Warehousing

COURSE CODE: MCA17508DCE

Metadata

An important component of the data warehouse environment is Meta data. Meta data, or data about data, has been a part of the information processing milieu for as long as there have been programs and data. But in the world of data warehouses, Meta data takes on a new level of importance, for it affords the most effective use of the data warehouse. Meta data allows the end user/DSS analyst to navigate through the possibilities. Put differently, when a user approaches a data warehouse where there is no Meta data, the user does not know where to begin the analysis. The user must poke and probe the data warehouse to find out what data is there and what data is not there and considerable time is wasted. Even after the user pokes around, there is no guarantee that he or she will find the right data or correctly interpret the data encountered. With the help of Meta data, however, the end user can quickly go to the necessary data or determine that it isn't there. Meta data then acts like an index to the contents of the data warehouse. It sits above the warehouse and keeps track of what is where in the warehouse. Typically, items the Meta data store tracks are as follows:

- Structure of data as known to the programmer
- Structure of data as known to the DSS analyst
- Source data feeding the data warehouse
- Transformation of data as it passes into the data warehouse
- Data model
- Relationship between the data model and the data warehouse
- History of extracts

It is noteworthy that metadata comes in different varieties. One set of flavors that metadata comes in is that of *business metadata versus technical metadata*. Business metadata is that metadata that is of use and value to the business person. Technical metadata is that metadata that is of use and value to the technician.

Typically, the technical Meta data that describes the data warehouse contains the following:

- Data warehouse table structures
- Data warehouse table attribution
- Data warehouse source data (the system of record)
- Mapping from the system of record to the data
- Data model specification
- Extract logging
- Common routines for access of data
- Definitions/descriptions of data
- Relationships of one unit of data to another

Dimensional Modelling

Information systems fall into two major categories: those that support the execution of business processes- Operational System and those that support the analysis of business processes-Analytic

Systems. The principles of dimensional design have evolved as a direct response to the unique requirements of analytic systems. The core of every dimensional model is a set of business metrics that captures how a process is evaluated, and a description of the context of every measurement.

Implemented in a relational database, the optimal schema design for an operational system is widely accepted to be one that is in third normal form. The design may be depicted as an entity-relationship model, or ER model. Coupled with appropriate database technology, this design supports high-performance inserting, updating, and deleting of atomic transactions in a consistent and predictable manner. Developers refer to the characteristics of transaction processing as the ACID properties—atomic, consistent, isolated, and durable.

Interaction with an analytic system takes place exclusively through queries that retrieve data about business processes; information is not created or modified. These queries can involve large numbers of transactions, in contrast to the operational system's typical focus on individual transactions. Specific questions asked are less predictable, and more likely to change over time. Historic data will remain important to the analytic system long after its operational use has passed. The principles of dimensional modeling address the unique requirements of analytic systems. A dimensional design is optimized for queries that may access large volumes of transactions, not just individual transactions. It is not burdened with supporting concurrent, high-performance updates. It supports the maintenance of historic data, even as the operational systems change or delete information. Dimensional modeling is widely accepted as the preferred technique for presenting analytic data because it addresses two simultaneous requirements:

■ Deliver data that's understandable to the business users.

■ Deliver fast query performance.

Dimensional modeling is a longstanding technique for making databases simple. In case after case, for more than five decades, IT organizations, consultants, and business users have naturally gravitated to a simple dimensional structure to match the fundamental human need for simplicity. Simplicity is critical because it ensures that users can easily understand the data, as well as allows software to navigate and deliver results quickly and efficiently.

Although dimensional models are often instantiated in relational database management systems, they are quite different from third normal form (3NF) models which seek to remove data redundancies. Normalized 3NF structures divide data into many discrete entities, each of which becomes a relational table. A database of sales orders might start with a record for each order line but turn into a complex spider web diagram as a 3NF model, perhaps consisting of hundreds of normalized tables.

The industry sometimes refers to 3NF models as entity-relationship (ER) models. Entity-relationship diagrams (ER diagrams or ERDs) are drawings that communicate the relationships between tables. Both 3NF and dimensional models can be represented in ERDs because both consist of joined relational tables; the key difference between 3NF and dimensional models is the degree of normalization.

Because both model types can be presented as ERDs, we refrain from referring to 3NF models as ER models; instead, we call t hem normalized models to minimize confusion.

Normalized 3NF structures are immensely useful in operational processing because an update or insert transaction touches the database in only one place. Normalized models, however, are too complicated for BI queries. Users can't understand, navigate, or remember normalized models that resemble a map of the Los Angeles freeway system. Likewise, most relational database management systems can't efficiently query a normalized model; the complexity of users' unpredictable queries overwhelms the database optimizers, resulting in disastrous query performance. The use of normalized modeling in the DW/BI presentation area defeats the intuitive and high-performance retrieval of data. Fortunately, dimensional modeling addresses the problem of overly complex schemas in the presentation area.

A dimensional model contains the same information as a normalized model, but packages the data in a format that delivers user understandability, query performance, and resilience to change.

The founding principle of dimensional design is disarmingly simple. Dimensional design supports analysis of a business process by modeling how it is measured.

Measurement is easy to discern, whether by listening to people talk or reading a report or chart. Consider the following business questions:

- What are gross margins by product category for January?
- What is the average account balance by education level?
- How many sick days were taken by marketing employees last year?
- What are the outstanding payables by vendor?
- What is the return rate by supplier?

Each of these questions centers on a business process: sales, account management, attendance, payables, return processing. These process-centric questions do not focus on individual activities or transactions. To answer them, it is necessary to look at a group of transactions.

Most importantly, each of these questions reveals something about how its respective business process is measured. The study of sales involves the measurement of gross margin. Financial institutions measure account balance. In human resources, they measure number of absences. The finance department measures payables. Purchasing managers watch the return quantities.

Without some kind of context, a measurement is meaningless. If you are told "sales were $10,000," there is not much you can do with this information. Is that sales of a single product, or many products? Does it represent a single transaction, or the company's total sales from conception to date? Without some context, the measurement is useless.

As with the measurements themselves, context is revealed in business questions or reports. In the preceding questions, for example, gross margin is viewed in the context of product categories and

time (the month of January). Sick days are viewed in the context of a department (marketing) and time (last year). Payables are viewed in the context of their status (outstanding) and vendor.

These two simple concepts, measurement and context, are the foundation of dimensional design. Every dimensional solution describes a process by capturing what is measured and the context in which the measurements are evaluated.

In a dimensional design, measurements are called *facts*, and context descriptors are called *dimensions*. Every dimensional design sorts information requirements into these categories. They may be identified within statements or questions, or found within report specifications. Sorting them into categories for facts and dimensions is easy, once you know what to look for.

A dimensional design organizes facts and dimensions for storage in a database management system. In a relational database management system (RDBMS), the design is referred to as a star schema. In a multidimensional database (MDB), the design is referred to as a cube.

Star Schemas versus OLAP Cubes

Dimensional models implemented in relational database management systems are referred to as star schemas because of their resemblance to a star-like structure. Related dimensions are grouped as columns in dimension tables, and the facts are stored as columns in a fact table. The star schema gets its name from its appearance: when drawn with the fact table in the center, it looks like a star or asterisk.

Dimensional models implemented in multidimensional database environments are referred to as online analytical processing (OLAP) cubes, as illustrated in Figure 1-1. If your DW/BI environment includes either star schemas or OLAP cubes, it leverages dimensional concepts. Both stars and cubes have a common logical design with recognizable dimensions; however, the physical implementation differs.

When data is loaded into an OLAP cube, it is stored and indexed using formats and techniques that are designed for dimensional data. Performance aggregations or precalculated summary tables are often created and managed by the OLAP cube engine. Consequently, cubes deliver superior query performance because of the precalculations, indexing strategies, and other optimizations. Business users can drill down or up by adding or removing attributes from their analyses with excellent performance without issuing new queries. OLAP cubes also provide more analytically robust functions that exceed those available with SQL. The downside is that you pay a load performance price for these capabilities, especially with large data sets.

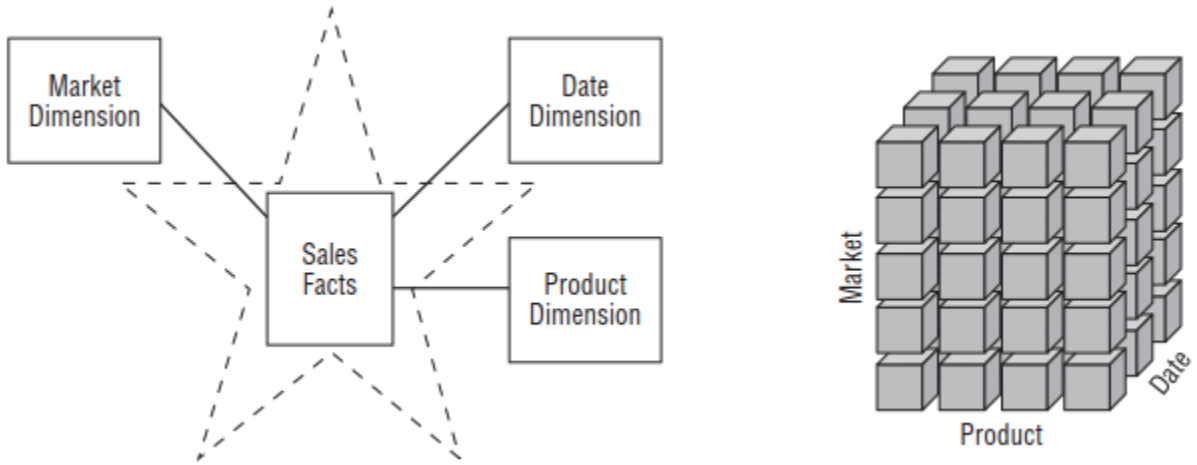


Figure 1-1 Star Schema Versus OLAP Cube

Key Components of Star Schema

Fact Tables for Measurements

The fact table in a dimensional model stores the performance measurements resulting from an organization's business process events. You should strive to store the low-level measurement data resulting from a business process in a single dimensional model. Because measurement data is overwhelmingly the largest set of data, it should not be replicated in multiple places for multiple organizational functions around the enterprise. Allowing business users from multiple organizations to access a single centralized repository for each set of measurement data ensures the use of consistent data throughout the enterprise.

The term fact represents a business measure. Imagine standing in the marketplace watching products being sold and writing down the unit quantity and dollar sales amount for each product in each sales transaction. These measurements are captured as products are scanned at the register, as illustrated in Figure 1-2. Each row in a fact table corresponds to a measurement event. The data on each row is at a specific level of detail, referred to as the grain, such as one row per product sold on a sales transaction. One of the core tenets of dimensional modeling is that all the measurement rows in a fact table must be at the same grain. Having the discipline to create fact tables with a single level of detail ensures that measurements aren't inappropriately double-counted.

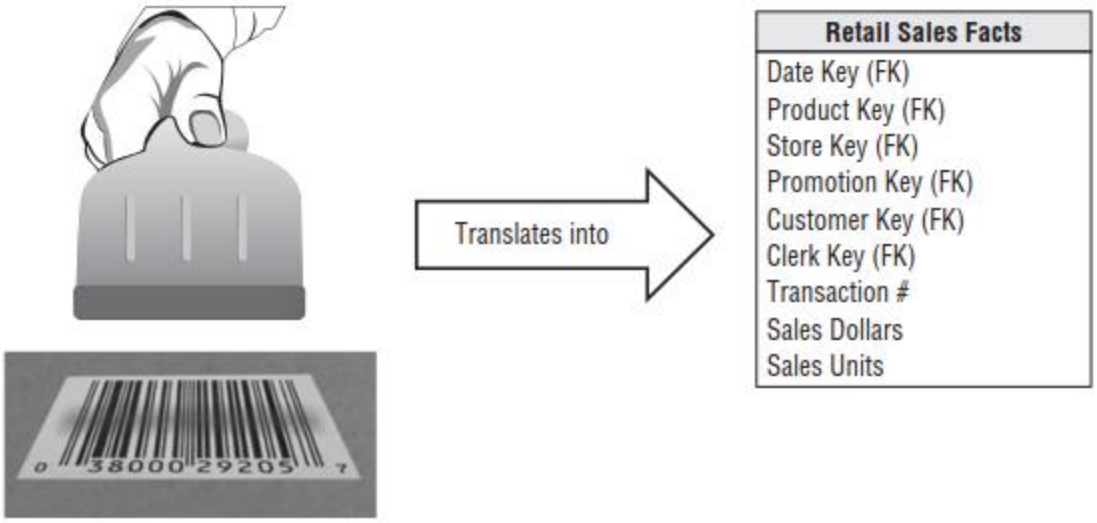


Figure 1.2 : Business process measurement events translate into fact tables.

The most useful facts are *numeric and additive*, such as dollar sales amount. Additivity is crucial because BI applications rarely retrieve a single fact table row. Rather, they bring back hundreds, thousands, or even millions of fact rows at a time, and the most useful thing to do with so many rows is to add them up. No matter how the user slices the data in Figure 1-2, the sales units and dollars sum to a valid total. You will see that facts are sometimes semi-additive or even non-additive. Semi-additive facts, such as account balances, cannot be summed across the time dimension. Non-additive facts, such as unit prices, can never be added. You are forced to use counts and averages or are reduced to printing out the fact rows one at a time—an impractical exercise with a billion-row fact table. Facts are often described as continuously valued to help sort out what is a fact versus a dimension attribute. The dollar sales amount fact is continuously valued in this example because it can take on virtually any value within a broad range. It is theoretically possible for a measured fact to be textual; however, the condition rarely arises. In most cases, a textual measurement is a description of something and is drawn from a discrete list of values. The designer should make every effort to put textual data into dimensions where they can be correlated more effectively with the other textual dimension attributes and consume much less space. You should not store redundant textual information in fact tables. Unless the text is unique for every row in the fact table, it belongs in the dimension table.

The fact table generally has its own primary key composed of a subset of the foreign keys. This key is often called a composite key. Every table that has a composite key is a fact table. Fact tables express many-to-many relationships. All others are dimension tables.

Dimension Tables for Descriptive Context

Dimension tables are integral companions to a fact table. The dimension tables contain the textual context associated with a business process measurement event. They describe the "who, what, where, when, how, and why" associated with the event.

As illustrated in Figure 1-3, dimension tables often have many columns or attributes. It is not uncommon for a dimension table to have 50 to 100 attributes; although, some dimension tables naturally have only a handful of attributes.

Dimension tables tend to have fewer rows than fact tables, but can be wide with many large text columns. Each dimension is defined by a single primary key (refer to the PK notation in Figure 1-3), which serves as the basis for referential integrity with any given fact table to which it is joined.

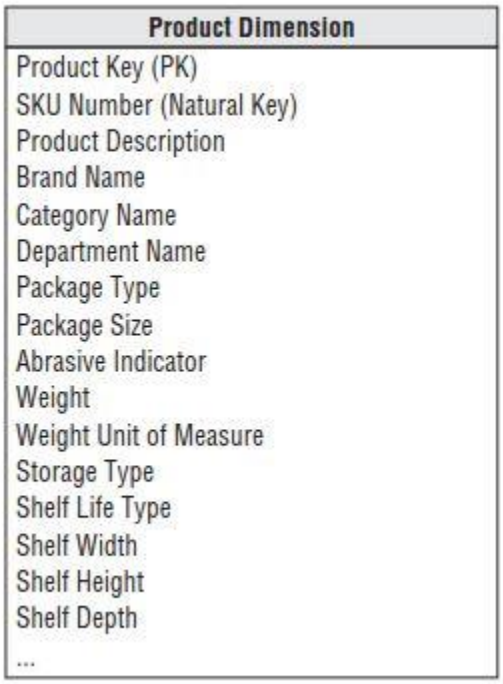


Figure 1-3: Dimension tables contain descriptive characteristics of business process nouns.

Dimension attributes serve as the primary source of query constraints, groupings, and report labels. In a query or report request, attributes are identified as the by words. For example, when a user wants to see dollar sales by brand, brand must be available as a dimension attribute.

Dimension table attributes play a vital role in the DW/BI system. Because they are the source of virtually all constraints and report labels, dimension attributes are critical to making the DW/BI system usable and understandable. Attributes should consist of real words rather than cryptic abbreviations. You should strive to minimize the use of codes in dimension tables by replacing them with more verbose textual attributes.

Figure 1-4 shows that dimension tables often represent hierarchical relationships. For example, products roll up into brands and then into categories. For each row in the product dimension, you should store the associated brand and category description. The hierarchical descriptive information is stored redundantly in the spirit of ease of use and query performance. You should resist the perhaps habitual urge to normalize data by storing only the brand code in the product dimension and creating a separate brand lookup table, and likewise for the category description in a separate category lookup table. This normalization is called snowflaking. Instead of third normal

form, dimension tables typically are highly denormalized with flattened many-to-one relationships within a single dimension table. Because dimension tables typically are geometrically smaller than fact tables, improving storage efficiency by normalizing or snowflaking has virtually no impact on the overall database size. You should almost always trade off dimension table space for simplicity and accessibility.

Product Key	Product Description	Brand Name	Category Name
1	PowerAll 20 oz	PowerClean	All Purpose Cleaner
2	PowerAll 32 oz	PowerClean	All Purpose Cleaner
3	PowerAll 48 oz	PowerClean	All Purpose Cleaner
4	PowerAll 64 oz	PowerClean	All Purpose Cleaner
5	ZipAll 20 oz	Zippy	All Purpose Cleaner
6	ZipAll 32 oz	Zippy	All Purpose Cleaner
7	ZipAll 48 oz	Zippy	All Purpose Cleaner
8	Shiny 20 oz	Clean Fast	Glass Cleaner
9	Shiny 32 oz	Clean Fast	Glass Cleaner
10	ZipGlass 20 oz	Zippy	Glass Cleaner
11	ZipGlass 32 oz	Zippy	Glass Cleaner

Figure 1-4: Sample rows from a dimension table with denormalized hierarchies.

Facts and Dimensions Joined in a Star Schema

Now that you understand fact and dimension tables, it's time to bring the building blocks together in a dimensional model, as shown in Figure 1-5. Each business process is represented by a dimensional model that consists of a fact table containing the event's numeric measurements surrounded by a halo of dimension tables that contain the textual context that was true at the moment the event occurred. This characteristic star-like structure is often called a star join, a term dating back to the earliest days of relational databases.

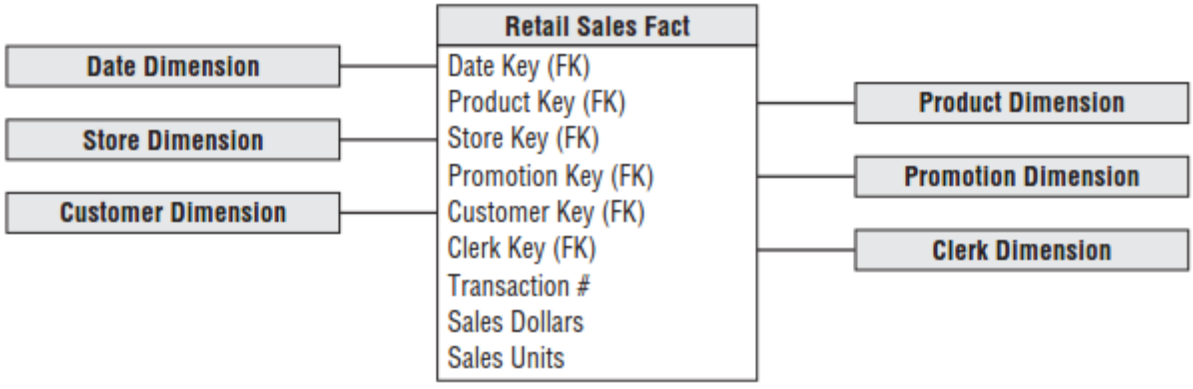


Figure 1-5: Fact and dimension tables in a dimensional model.

The simplicity of a dimensional model also has performance benefits. Database optimizers process these simple schemas with fewer joins more efficiently. A database engine can make strong assumptions about first constraining the heavily indexed dimension tables, and then attacking the fact table all at once with the Cartesian product of the dimension table keys satisfying the user's constraints.

Amazingly, using this approach, the optimizer can evaluate arbitrary n-way joins to a fact table in a single pass through the fact table's index.

Basic Dimension Table Techniques

A well-developed set of dimension tables provides powerful and diverse analytic capabilities. The dimensions provide contextual information, without which reports would be meaningless. Successful dimension design hinges on the proper use of keys, the development of a richly detailed set of dimension columns, and a rejection of the urge to save space.

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, fl at 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.

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.

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

In some cases, it can be useful to create a table that contains dimensions that do not have any real relationship to one another. The orders schema shown in Figure 1.6, for example, might benefit from the addition of several attributes to describe the type of order being placed, whether it was a credit order, whether it was solicited, and whether it represents a reorder. While these various indicators do not relate directly to one another, they can be combined into a single table for convenience. The result is a junk dimension, as depicted in Figure 1.6.

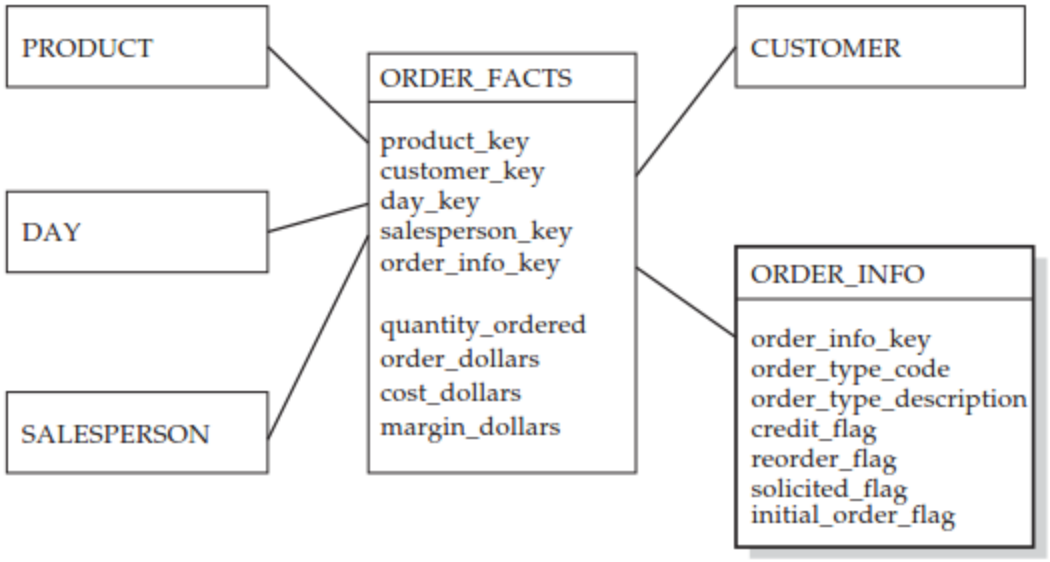


Figure 1.6: A junk dimension table collects unrelated dimensions for convenience

A junk dimension like the order_info table in Figure 1.6 has no natural key. It can be built by populating it with each possible combination of column values (a Cartesian product). When facts are loaded, they are associated with the row in this table that corresponds to the appropriate values of the various columns.

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.

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 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.

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, 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.

Figure 1.6 depicts four facts: quantity_ordered, order_dollars, cost_dollars, and margin_dollars. Each of these facts is fully additive; they may be summed up across any and all of the dimensions in the schema, producing a meaningful result. Not every measurement exhibits additivity. Many key business metrics are expressed as rates or percentages. This type of measurement is never additive. For example,

Table 1.1 shows the margin rate on each of the orders for “Gel Pen Red” on January 1.

Day	Salesperson	Product	Customer	Margin Rate
1/1/2009	Jones	Gel Pen Red	Balter Inc.	3.02%
1/1/2009	Jones	Gel Pen Red	Raytech	3.02%
1/1/2009	Baldwin	Gel Pen Red	Venerable Holdings	3.02%
1/1/2009	Baldwin	Gel Pen Red	eMart LLC	3.02%
1/1/2009	Baldwin	Gel Pen Red	Shatter & Lose	3.02%
1/1/2009	Sebenik	Gel Pen Red	Comstock Realty	3.02%
1/1/2009	Sebenik	Gel Pen Red	RizSpace	3.02%
1/1/2009	Sebenik	Gel Pen Red	StarComp	3.02%
1/1/2009	Sgamma	Gel Pen Red	Implosion Town	3.02%
			Sum:	**27.18%**

Table 1.1 Margin Rates Cannot Be Added Together

The margin rate is the percentage of the sale that represents profit, and it is closely monitored by management. In this table, each individual transaction has a margin rate of 3.02 percent. It is not possible to summarize the margin rate for these transactions by adding them together.

This would produce a margin rate of over 27 percent, which is clearly incorrect. Luckily, there is a solution. Ratios can be broken down into underlying components that are additive. In this case, the margin rate is the ratio of the margin dollars to order dollars. These components are fully additive. They can be stored in a fact table and safely aggregated to any level of detail within a query or report. An example report is shown in Figure 1.7. The nonadditive fact margin_rate is not stored in the fact table; it is computed as the ratio of the sums of margin dollars and order dollars. This computation may be done in a query or by additional processing logic in the reporting tool. Care must be taken with subtotals and grand totals in the report; the margin rate in these rows must be computed as the ratio of the subtotals for margin dollars and order dollars.

While nonadditive facts are not stored in fact tables, it is important not to lose track of them. For many processes, ratios are critical measurements without which a solution would leave much to be desired.

Margin Report

Date: January 1, 2009
Product: Gel Pen Red

Salesperson	Customer	Margin Dollars	Order Dollars	Margin Rate
Jones	Balter Inc.	192.74	6,382.21	3.02%
	Raytech	39.05	1,293.11	3.02%
	Subtotal:	*231.79*	*7,675.32*	*3.02%*
Baldwin	Venerable Holdings	121.50	4,023.22	3.02%
	eMart LLC	253.44	8,392.00	3.02%
	Shatter & Lose	8.74	289.54	3.02%
	Subtotal:	*383.68*	*12,704.76*	*3.02%*
Sebenik	Comstock Realty	12.06	399.29	3.02%
	RizSpace	58.10	1,923.93	3.02%
	Starcomp	90.36	2,992.11	3.02%
	Subtotal:	*160.52*	*5,315,33*	*3.02%*
Sgamma	Implosion Town	213.88	7,082.22	3.02%
	DemiSpace	113.92	3,772.11	3.02%
	Subtotal:	*327.80*	*10,854.33*	*3.02%*
Grand Total:		***1,103.80***	***36,549.74***	***3.02%***

Margin Rate is a nonadditive fact.

Summary row is computed as a ratio of the subtotals, not by summing margin rates for the salesperson.

Figure 1.7 Nonadditive facts are computed as the ratio of additive facts

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 OLAP 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.

Degenerate Dimensions

Sometimes, it is not possible to sort all the dimensions associated with a business into a neat set of tables. In situations like this, it may be appropriate to store one or more dimensions in the fact table. When this is done, the dimension column is called a degenerate dimension. Although stored in the fact table, the column is still considered a dimension. Like the dimension columns in other tables, its values can be used to filter queries, control the level of aggregation, order data, define master–detail relationships, and so forth. Degenerate dimensions should be used cautiously. Because the fact table accumulates rows at a rapid pace, the inclusion of degenerate dimensions can lead to an excessive consumption of space, particularly for textual elements. In most cases, candidates for degenerate dimensions are better placed in junk dimensions. Transaction identifiers are exceptions to this guideline.

Transaction identifiers are commonly stored as degenerate dimensions. They may also serve as a unique identifier for fact table rows, and define fact table grain. The orders star in Figure 1.6 was criticized for not storing granular data. It can be redesigned to store information at the order line level of detail by adding degenerate dimensions that identify the order and order line. The result is shown in Figure 1.8 .The grain of the fact table in Figure 1.8 can be stated as “orders at the order line level of detail.” This has been achieved by adding transaction identifiers from the source system to identify discrete order lines: the order_id and order_line. Together, these two attributes can serve as a unique identifier for fact table rows.

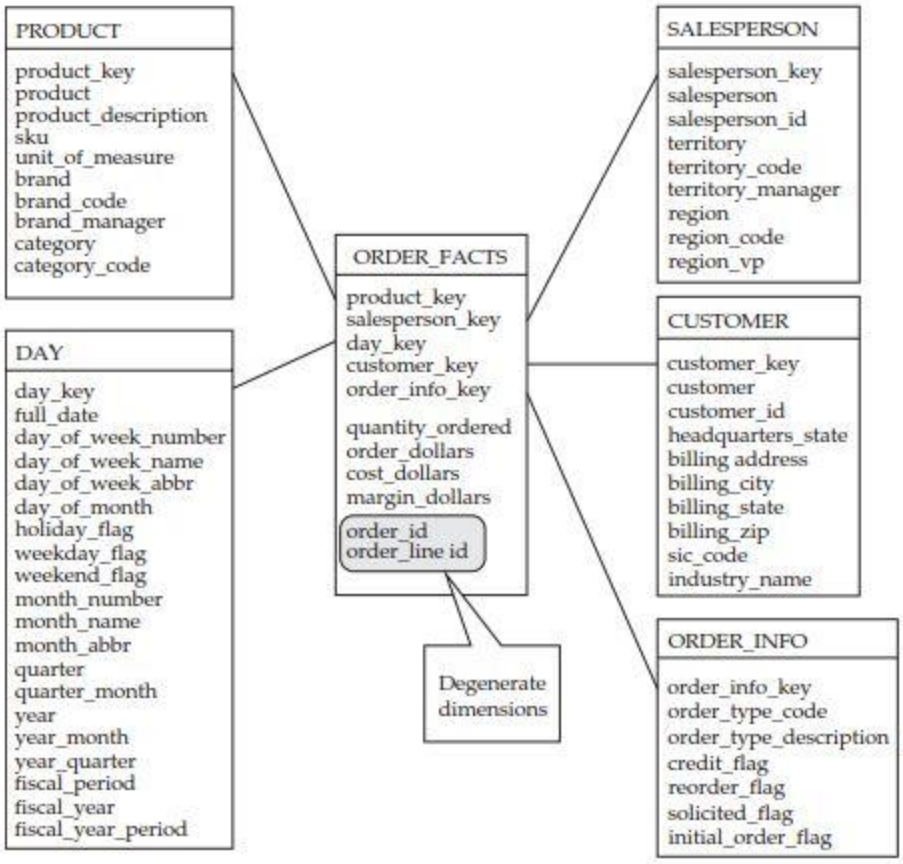
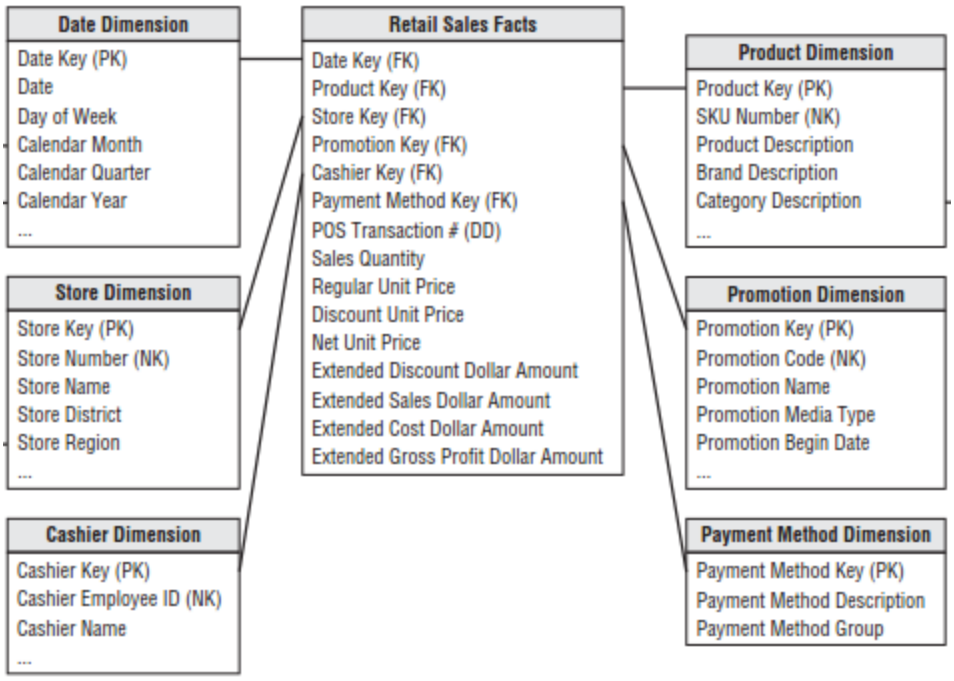


Figure 1.8 Degenerate dimensions define the grain of this fact table

Retail Sales Schema

A star schema for Retail Sales is given below, POS transaction number is the degenerate dimension (identified by the DD notation), the natural keys are identified with the NK. The natural key is often modeled as an attribute in the dimension table. If the natural key comes from multiple sources, you might use a character data type that prepends a source code, such as SAP|43251 or CRM|6539152. If the same entity is represented in both operational source systems, then you'd likely have two natural key attributes in the dimension corresponding to both sources.



Retail Sales Schema

Slowly Changing Dimensions

In every dimensional design, it is crucial to identify how changes in source data will be represented in dimension tables. This phenomenon is referred to as slowly changing dimensions. This term gets its name from the relatively slow rate at which dimensions accumulate changes, at least when compared with fact tables, which accumulate rows at a rapid pace. A variety of responses to changed data elements are possible. In some cases, there may be no analytic value in preserving history. In other cases, it may be critical that historic data be maintained.

Figure 1.9 shows a customer record in an order entry system at three different points in time. The record in question is that of customer_id 9900011, which happens to be someone named Sue Johnson. Notice that on January 1, 2007, her date of birth is indicated as March 2, 1961, and she lives in the state of Arizona (AZ). Later, her date of birth has been changed from a date in 1961 to 1971. Still later, this same customer has moved; her state of residence is now California (CA). This operational system has handled both changes in the same way: by overwriting the record for customer_id 9900011.Suppose that this operational system feeds a dimension table in a star schema that tracks orders. Analytic requirements may call for the changes to be treated differently. In the case of Sue's date of birth, the business may consider the change history to be insignificant.

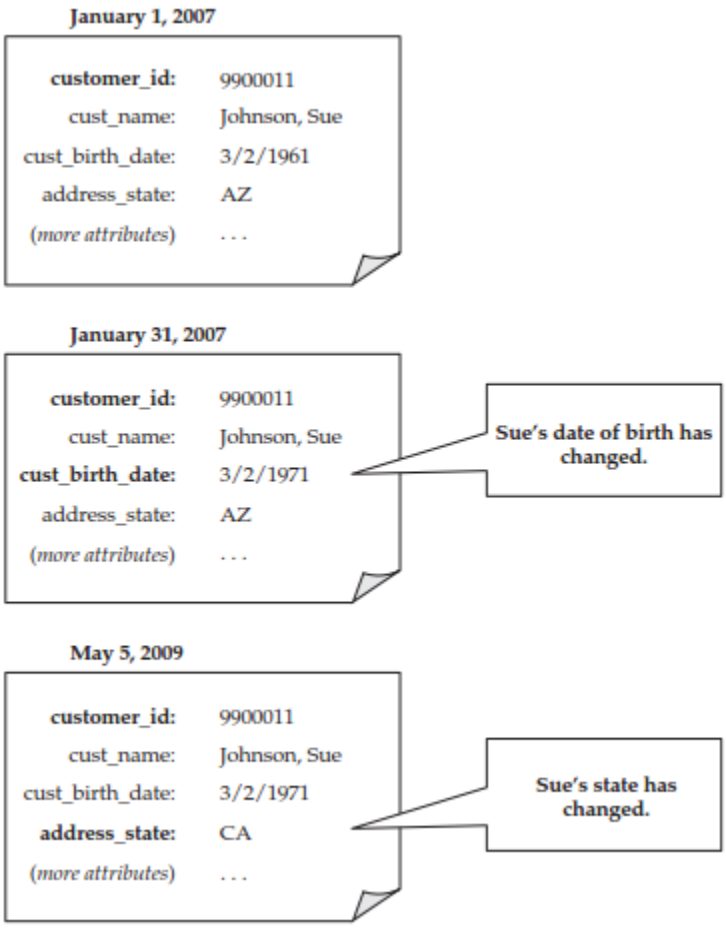


Figure 1.9 Changes in operational data

As in the operational system, it may be best simply to update the record for Sue in a customer dimension table. Sue's address change, on the other hand, may have more significance. She may have placed some orders while living in Arizona and some more orders while living in California. If someone is studying order history by state, it is important that each of Sue's orders be associated with the correct state. These two changes to customer 9900011 can be used to illustrate the most common types of slow change response in a star schema: the type 1 slow change and the type 2 slow change. These responses overwrite transaction history and preserve it, respectively. There is also a less common type 3 response. It is used in situations that do not require historic context but do call for use of both the before and after values of the changed data element.

Type 1 Change

When the source of a dimension value changes, and it is not necessary to preserve its history in the star schema, a type 1 response is employed. The dimension is simply overwritten with the new value. This technique is commonly employed in situations where a source data element is being changed to correct an error.

By overwriting the corresponding dimension in the star schema, the type 1 change obliterates the history of the data element. The star carries no hint that the column ever contained a different value. While this is generally the desired effect, it can also lead to confusion. If there were any associated facts before the change occurred, their historic context is retroactively altered.

Overwriting the Dimension Value

Recall, for example, the change in date of birth for Sue Johnson. According to Figure 1.9, Sue Johnson was initially recorded as having a birth date in 1961. Later, her date of birth was updated to show she was born in 1971. People's birth dates do not change, so when this information was updated in the source system, it was presumably the correction of an error.

Assume that a star schema has been developed to track the orders process, with a dimension table called customer. This table carries the customer_id as a natural key, and also has columns for the customer's name, date of birth, and state of residence. The top portion of Figure 1.10 shows the state of affairs in the star schema before any changes have occurred.

In the customer dimension table, there is a row for Sue, which is highlighted. You can see that this row contains the customer_id 990001. This is a natural key column; it identifies the record for Sue in the source system. There is also a surrogate key called customer_key, which contains the value 1499. Sue's date of birth is shown as 3/2/1961, and her state is shown as AZ. Presumably, there are numerous other columns in the customer table that are not shown in the picture.

Just below this table, still in the top half of Figure 1.10, a row from a fact table is shown. The customer_key in this row is 1499, which refers to Sue's record in the customer dimension table. You can interpret this row as follows: on the date represented by day_key 2322, Sue bought five units of whatever product is represented by product_key 10119. One would not ordinarily be studying key values, but the corresponding tables have been omitted to keep the diagram simple.

The bottom half of Figure 1.10 shows what the star schema looks like after a type 1 change to Sue's date of birth occurs. The row in the customer dimension table for customer_id 9900011 has been updated; Sue's date of birth is now shown as 3/2/1971. This row is still represented by the surrogate key 1499.

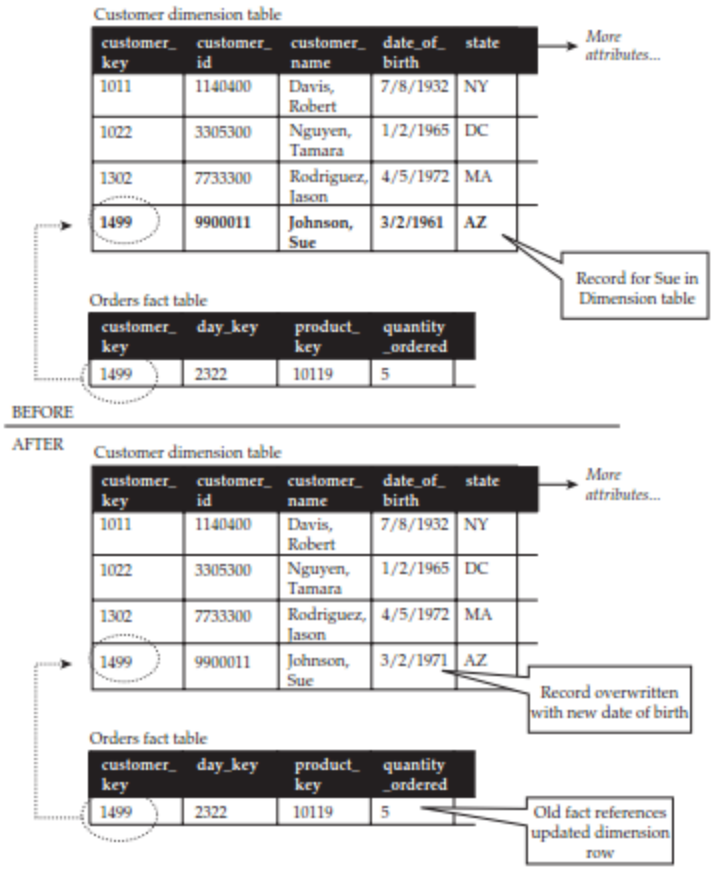


Figure 1.10: A type 1 change: before and after

Preexisting Facts Have a New Context

A type 1 change has an important effect on facts, one that is often overlooked. When a record is updated in a dimension table, the context for existing facts is restated. This effect can give rise to confusion.

An example is provided by the change in Sue's date of birth. The "before" picture in the top half of Figure 1.10 shows an order in the fact table for Sue. It was placed on whatever day is represented by day_key 2322. If you were to run a query that grouped orders for that day by the customer's date of birth, Sue's order would be grouped together with other customers born on 3/2/1961. Suppose that someone created a report to do just that and printed out the results.

Now, move forward to a point in time after the type 1 change has taken place. The bottom half of Figure 1.10 shows that Sue is now listed with a date of birth of 3/2/1971. You run the same report—taking all orders from the day represented by day_key 2322 and grouping them by customers' birth dates. Your report will count Sue's past order with people born on 3/2/1971, while the one printed previously will count it among people born on 3/2/1961. The reports have different figures, even though they both describe activity for the same day in the past.

History of Dimension Is Not Maintained

In addition to restating the context of facts, the type 1 change fails to track the history of the dimension itself. No change history of the data element is maintained in the star schema. In a Corporate Information Factory architecture, the data warehouse may maintain this history in the enterprise data warehouse repository.

Type 2 Change

Most operational changes are dealt with in a star schema as type 2 changes. The type 2 change preserves the history of facts. Facts that describe events before the change are associated with the old value; facts that describe events after the change are associated with the new value.

Inserting a New Dimension Row

The second method for responding to a change in source data is to insert a new record into the dimension table. Any previously existing records are unchanged. This type 2 response preserves context for facts that were associated with the old value, while allowing new facts to be associated with the new value.

Sue Johnson's change of address provides an example where a type 2 change can be useful. Recall from Figure 1.9 that over the years, Sue has lived in Arizona, and later in California. She may have placed some orders while living in Arizona and other orders while living in California. A type 1 response to Sue's change in address would have the undesirable side effect of restating the context for orders that Sue placed before she moved. They would become associated with California, even though Sue lived in Arizona at the time.

Figure 2.0 illustrates a type 2 response to Sue's change of address. In the "before" section of this diagram, there is a record in the dimension table that shows customer_id 9900011 (Sue) as residing in the state of Arizona (AZ). This row has a surrogate key value of 1499. A row in the fact table contains this key value, indicating that Sue has placed an order.

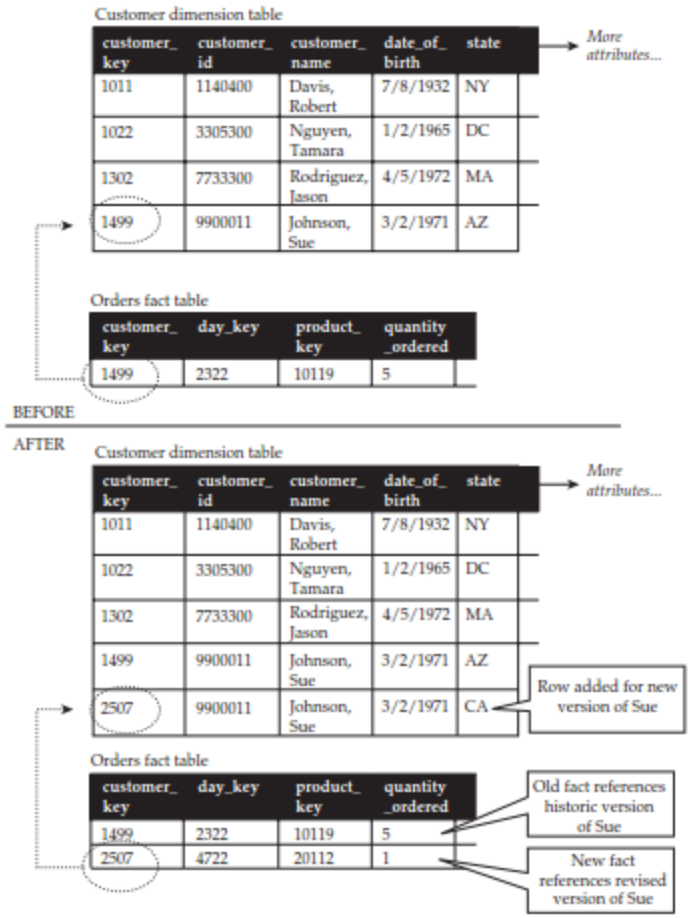


Figure 2.0: A type 2 change: before and after

The lower half of Figure 2.0 shows what happens when Sue's address changes. In the dimension table, the preexisting row for Sue is left untouched. It still shows Sue as residing in Arizona. A new record has been added for Sue. This record carries surrogate key 2507. This new row indicates that customer 9900011 is Sue Johnson and that she lives in the state of California (CA).

This type 2 response has the effect of creating "versions" of Sue in the dimension table. Where there was previously one row representing Sue, there are now two. This "versioning" is made possible because the dimension table does not rely on the natural key, customer_id, as its unique identifier.

Historic Context of Facts Is Preserved

By creating multiple versions of the dimension, a type 2 response avoids restating the context of previously existing facts. Old facts can remain associated with the old row; new facts can be associated with the new row. This has the desired effect of preserving past history, while allowing new activity to be associated with the new value.

To understand how the type 2 change preserves history, look at the fact table in the lower half of Figure 2.0. After the type 2 change has occurred, the order placed before Sue moved remains associated with the "old" version of Sue. You can use the foreign key value in the fact table to

verify this. Tracing customer_key 1499 to the customer table leads to a row that identifies customer 9900011 as Sue Johnson, living in Arizona. The fact table now contains a new row, this one for an order placed after the move. The new row carries the surrogate key 2507, which refers to the "new" version of Sue, living in California. Any reports that group orders by state will continue to group Sue's old order with "AZ," while her new order will be grouped with "CA."

When a type 2 change occurs, not all dimension values will be altered. For example, after Sue moved, her date of birth remained the same: 3/2/1971. Any query that groups orders by date of birth will associate all of Sue's orders with that date. If someone wants to look at all orders associated with Sue, they can group all orders for her customer_id, which is 9900011.

History of Dimension Is Partially Maintained

A type 2 change results in multiple dimension rows for a given natural key. While this serves to preserve the historic context of facts, it can trigger new forms of confusion. Users may be confused by the presence of duplicate values in the dimension tables. Designers may be lulled by a false sense that they are preserving dimensional history.

Type 2 changes can confuse end users because they cause duplicate values to appear in dimension tables. For example, after Sue Johnson's change of address, there are two rows in the dimension table for her customer_id. If someone were to query the dimension table to get the name associated with customer_id 9900011, both rows would be returned. This side effect can be avoided by issuing browse queries that select distinct values.

Although the type 2 change preserves the historic context of facts, it does not preserve history in the dimension. It is easy to see that a given natural key has taken on multiple representations in the dimension, but we do not know when each of these representations was correct. This information is only provided by way of a fact.

For example, after the change to Sue's address has occurred, the dimension table in Figure 2.0 shows that there have been two versions of Sue, but it cannot tell us what Sue looked like on any given date. Where was she living on January 1, 2008? The dimension table does not carry this information. If there is an order for January 1, 2008, we are in luck, because the orders fact table will refer to the version of Sue that was correct at the time of the order. If there is not an order on that date, we are unable to determine what Sue looked like at that point in time.

It may be clear to you that this problem is easily rectified by adding a date stamp to each version of Sue. This technique allows the dimension to preserve both the history of facts and the history of dimensions

Cubes

Dimensional models are not always implemented in relational databases. A multidimensional database, or MDB, stores dimensional information in a format called a cube. The basic concept behind a cube is to precompute the various combinations of dimension values and fact values so they can be studied interactively.

Multidimensional Storage vs. Relational Storage

The primary advantage of the multidimensional database is speed. A cube allows users to change their perspective on the data interactively, adding or removing attributes to or from their view and receiving instantaneous feedback. This process is often referred to as Online Analytical Processing, or OLAP. OLAP interaction with a cube is highly responsive; there is instantaneous feedback as you slice and dice, drill up and drill down. In contrast, a star schema is interacted with through a query-and-response paradigm. Each change in the information detail on display requires the issuance of a new query.

Another advantage of the multidimensional database is that it is not hampered by the limitations of SQL. Because it specializes in the storage of facts and dimensions, it can offer interfaces to ask for information that SQL does not traditionally handle well. MDBs were providing running totals, rankings and other statistical operations long before these capabilities were added to SQL. Multidimensional databases may also offer specialized support for recursive hierarchies, which may be ragged, something that requires a bridge table in the star schema world.

Of course, all this capability comes with a cost. As the number of dimensions and their values increase, the number of possible combinations that must be precomputed explodes. This limits the ability of the cube to scale with large volumes of data. Typical measures to stem this limitation invariably reduce some of the benefits offered by the cube.

Data in an MDB is accessed through an interface, which is often proprietary, although MDX has gained wide acceptance as a standard. Still, the ability to write queries in this environment is a skill that is not as widely available. In contrast, there is a large pool of information technology professionals who understand SQL, and a wider variety of reporting tools that support it. To some, this is another disadvantage of the MDB. Figure 2.1 summarizes the differences in these technologies.

	Data Structure	Access Language	Style of Interaction	Advantages
Relational Database	 Star Schema	Structured Query Language (SQL)	Query and response	Scalable Widely understood access language
Multi-dimensional Database	 Cube	Proprietary API or MDX	Interactive (OLAP)	Fast Expressive access language

These distinctions are beginning to fade

Figure 2.1 Alternative storage technologies for dimensional data

Snowflake Schema

Snowflake Schema is the extension of the star schema. It adds additional dimensions to it. As its name suggests, it looks like a snowflake. In this schema, the dimension tables are normalized i.e. data is split into additional tables. Splitting the table reduces redundancy and memory wastage. It has the hierarchical form of dimensional tables. The dimension and sub-dimension tables are associated with the primary and foreign keys in the fact table. It is easier to implement and uses less disk space. As it has multiple tables the performance of the query is reduced. More maintenance is required because there are more lookup tables.

Consider a refrigerator manufacturing company and we need to create a star schema for the sales of this refrigerator manufacturing company. Sales will have the following dimensions:

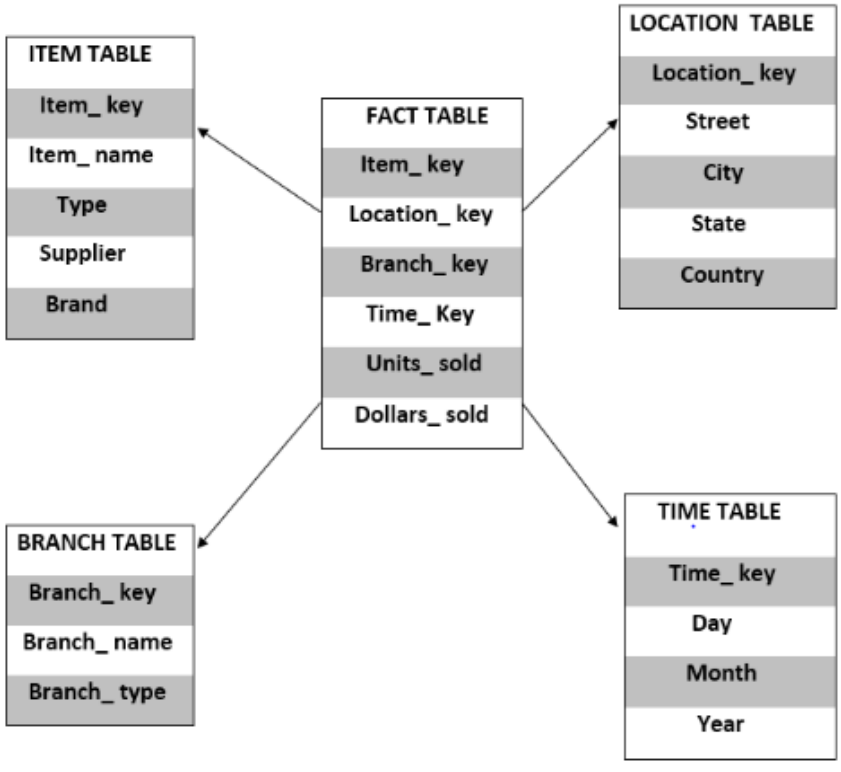
Item

Location

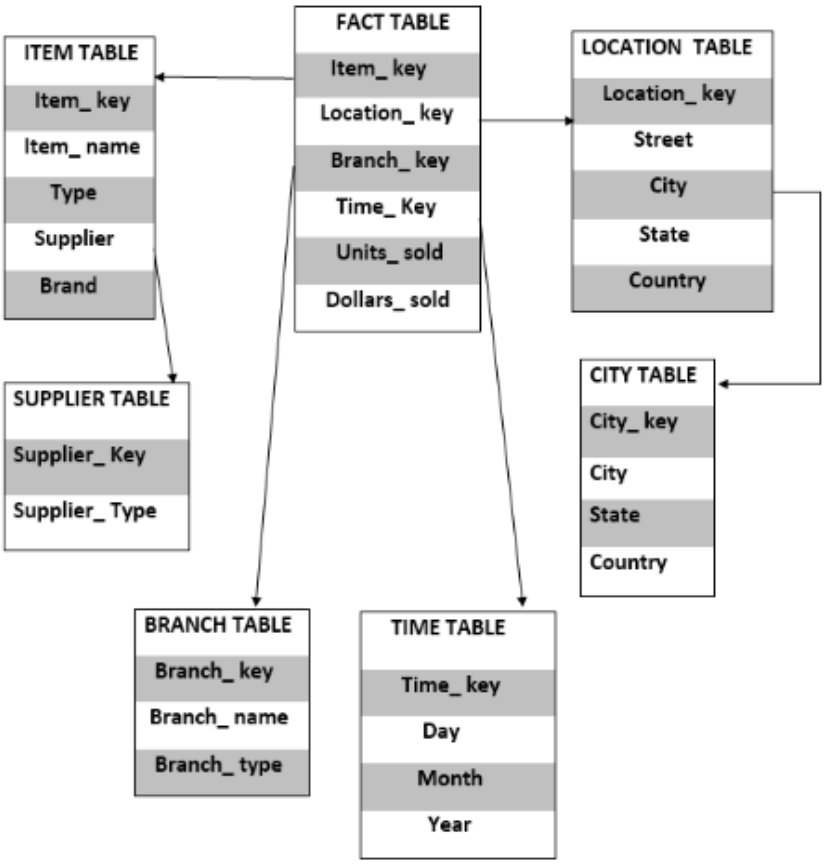
Branch

Time

The schema has a fact table at the center for sales which would contain keys to associate with each dimension, having two measures i.e. units sold and dollars sold.



In the snowflake schema the fact table is the same as in star schema but the major difference is in the definition or layout of dimension tables.



In this schema, the single dimension table of the item has been normalized and has been split and a new supplier table has been created including information on the type of supplier. Similarly, the dimension table of location is normalized and data is split into a new city table containing details of the particular city.

Further Reading

Paulraj Pooniah , " Data Warehousing Fundamentals " Wiley