

Building a Data Warehouse Ironbridge Software

MICHAEL J. DICKENSON



We Bridge Data



Index

Purpose	3
Content	3
Vision	4
Corporate Model	4
Data Model	5
Data Sources	6
Data Consistency and Harmonizing	6
Understanding Dimensional and Relational Data Models	7
Data Marts	16

Purpose

Your company has decided to build a Data Warehouse. Why? Has the CEO read about the marketing or product opportunities that can be unearthed with data mining? Or is it for the more prosaic reason that departments simply can't get data that works; the company has mountains of data but condensing it to be suitable for analysis or planning just takes too long.

The usual justification for a Data Warehouse is frustration in marketing or sales or finance or manufacturing or shipping (pick one). Frustration that the reports produced by the company's systems are inflexible; business conditions change all the time and it seems that users can never get their hands on exactly the right data to solve a problem or plan for an opportunity.

Users want to be able to pick and choose the data they need; they want different data sources integrated together; they want to use the data directly in a sales presentation or a business plan or an analysis quickly and easily.

One possible answer to solve this problem is to give business users better tools to access the company's transaction systems. I.T. departments are reluctant to do this, and rightly so. The company's transaction processing systems are designed and finely tuned to run the business: take orders, schedule manufacturing, ship product, cut invoices and so on. These systems can answer questions such as

"What is the status of my order?" *or*

"What is my account balance?"

not "Find all customers who have not yet ordered our new product line"

"Can we increase prices on some brands without losing customers?"

"How is northeast region performing compared to last year?"

An answer like this last one might require a complete roll-up of 2 years of financials all the way from individual customer invoices. It might work, but it would so badly affect the performance of the rest of the system, business would suffer.

So, the usual initial purpose of a Data Warehouse is simply to give end users more flexible access to data they already have. The DW grows from there. Eventually it will fulfill the CEO's dream of Data Mining: finding profitable customer demographics we have ignored, finding product mixes we have never tried and beyond.

Content

The next question to answer, knowing the purpose of the DW, is to decide what data goes in to it. Ideally, everything remotely to do with the business will be included; practically, we have to start somewhere short of that goal.

The **bottom-up** approach identifies the content of the enterprise's transaction processing systems (order entry, invoicing, accounting, personnel, shipping etc.); all the existing reporting systems; all desired and projected reporting requirements and finally what external data users may need (marketing research data from the Census Bureau for example). Then, we build an all-encompassing data model.

The **top-down** approach starts with a group of end users, identifies their most urgent needs and builds a smaller scale data model. Marketing may need consumer data from the syndicated data providers merged with product promotion data for an analysis; finance may want orders, invoices and costs for a profitability analysis.

The top-down approach is the most practical for two reasons:

1. The smaller scale is easier to deliver an end result quickly;
2. The clearly defined group of users are (or become) the project's sponsors; their needs come first and provide a sharper focus. If they are paying the bills the focus will be even sharper.

If you adopt the bottom-up approach, you will exhaust your funding and run out of management's patience and possibly lose your job before delivering an end result. A Data Warehouse is not a concept; it is a business application that must provide a return on the company's investment just like any other project. The Data

Warehouse project must be driven by user needs. "Build it and they will come" is not a DW strategy; it is a recipe for failure.

This end-user focus does not mean that the Data Warehouse is a collection of small reporting systems or 'data marts'. Even if the project starts small, it must be done with a vision for eventually serving all the company's data needs.

Vision

Whether using the top-down or bottom-up approach, the designers must have a long term vision for the design elements of the Data Warehouse. This vision must include:

- the corporate model
- the data model
- data sources
- data consistency and harmonizing
- data extracts, transforms and loading
- data validation
- reporting and analytical needs
- aggregation strategy
- sizing
- performance expectations
- hardware platform
- software platform
- growth requirements (both the growth of the data warehouse as data is collected overtime and the natural growth of the business)

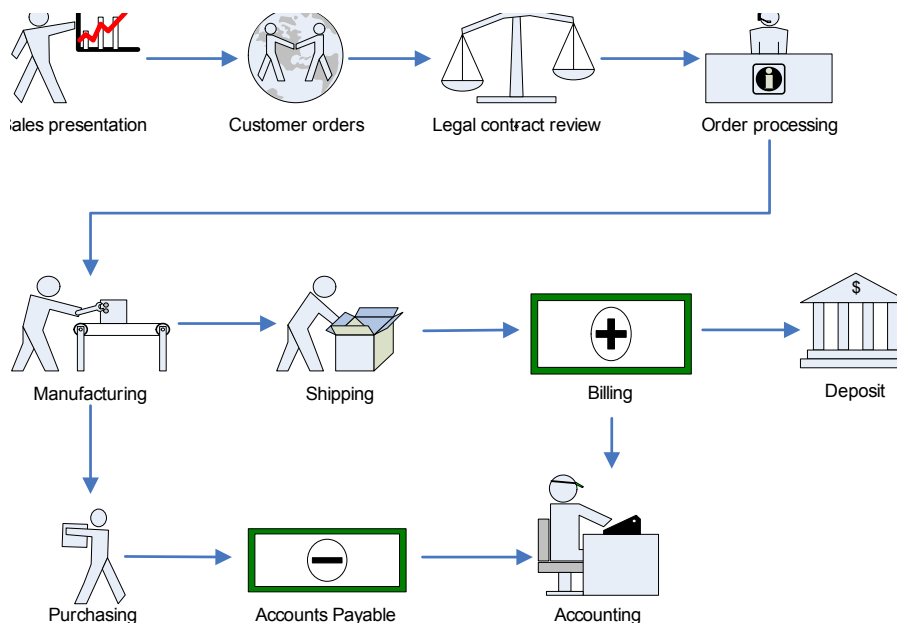
Corporate Model

The 'corporate model' consists of the functional entities in the company, the data elements they each use and the relationships between them. (This is also called the 'Functional Model')

Most industrial enterprises have functional entities such as sales, marketing, order processing, engineering, production, shipping, billing, accounting, purchasing, legal, human resources and so on. The first step is to identify the data elements each uses. These will be well known; the task is mostly to discover the correct names and structure of the elements.

Sales uses a 'customer number' or similar element; production and shipping use a product number or item number or stock keeping unit (SKU); accounting uses general ledger codes etc.

The relationship between the functional departments is also well known; making a simple process flow diagram will identify the relationships. Here is a simplistic example and in designing your corporate model yours should not be much more complex.



Remember to include the corporate calendar: does your company use 12 calendar months per year, or a weekly calendar with 4-4-5 weekly periods per quarter? If the calendar is weekly, does every seventh year require 53 weeks, or does the last or first week of the year stretch to accommodate the extra day? Does the week start at midnight Saturday? Some of these issues can be tricky to model in the Data Warehouse.

This 'corporate model' is simpler than the standard Business Process Model. When designing a data warehouse, we must concentrate on the data in the functions. Understanding the relationships and the flow of data is essential, of course, but we must avoid building them into the warehouse. This will limit its future flexibility.

Here's an example. Accounts Receivable is a standard function in the Finance Department. Finance uses the A/R system to manage cash, collections, write-offs and so on. It is a fairly self contained function. However, Marketing could use this information to detect a failing product; one that customers are delaying payment on for some reason. Sales could use this information to detect a sales rep. who is selling to unqualified accounts who end up not needing the product and not paying for it. ("Ice to Eskimos").

So, we must be sure that raw data such as ageing receivables can be linked all the way through the warehouse to the customer, the product purchased, the sales rep who sold it, the customer demographic, the date the product was manufactured etc..

When the corporate model is complete, the next step is to identify the *data elements* that each *functional entity* uses. The best way to do this is to track how the company's existing systems handle the flow of data between the functions.

The sales rep. who makes the sale has a number, so he/she can be paid commissions on the sale by the payroll department. The customer has a number so that shipping ships to the right place and billing sends the invoice to the right place (which may be different of course). The customer order was for a quantity of product which has a SKU, so that manufacturing knows what to make; shipping picks the right items from the warehouse and billing computes the bill correctly. Manufacturing places a purchase order with purchasing which flows to accounts payable when the supplier sends an invoice. Accounting needs to know all of these data elements to keep the books, to avoid paying any taxes and to publish the annual report.

Unless your company has completely re-engineered its systems (as many did to avoid Y2K problems) you will be astonished to find how inconsistent these data elements are between departments. This is a major challenge in Data Warehousing, to harmonize the data schemes within a company. The longer a company has been in business and the more acquisitions it has made over the years, the worse this problem grows.

The corporate model is the starting point for designing the Data Warehouse. It should NOT be exhaustive; it is the framework for moving forward. The most important end result of the model is a list of the data entities used, their structure (down as far as the number of digits in a customer number for example) and the needs for harmonizing.

Data Model

The Data Model is NOT a model of the functional entities in the company. Rather, it is the relationship between the data elements that resulted from the corporate model.

Remember that the purpose of a data warehouse is to facilitate access to cross-functional data, we want to break down the 'silos', 'chimneys', 'empires' that departments like to build for themselves. (In researching the corporate model, you will quickly find out who the silo builders, chimney erectors and pocket emperors are. They will be the least co-operative).

The data elements can almost always be assigned to one of 4 dimensions of data.

- 1) **Geography** – the company's regions, districts, territories, subsidiaries, plants, warehouses and ultimately customers. These are the physical entities that provide data to the transaction systems.
- 2) **Product** – the company's product lines and services. What customers pay for.

- 3) **Time** – usually data warehouses acquire data from their source systems daily or weekly. (It is unlikely that a DW captures data in real time – that is the function of transaction processing systems).
- 4) **Measures** or **Facts** – the data values the users want to see. Finance wants unit volume shipped and dollars invoiced; shipping wants costs.

Geography, product and time identify a business transaction:

- the who, what and when of a customer buying a product
- the who, what and when of an invoice being cut
- the who, what and when of a shipment being made

The why and the how questions are answered by attributes of these dimensions or sometimes by a 5th dimension. 'Why' a customer purchased is the most important answer a marketing analyst wants and involves trade promotion data, competitive activity and maybe manufacturing shortages.

Subsequent chapters describe sample data models and the different styles and products to support them.

Data Sources

The primary source of data for a Data Warehouse is the company's own transaction processing systems. Even if your company has re-engineered its systems with an ERP system (Enterprise Resource Planning) where all company functions are built into a consistent data structure, there will still be other sources to bring into the warehouse.

If you do not have the luxury of a sparkling new ERP system, it may be a challenge to identify all the sources of transaction data in the company. It will be a further challenge to build an automated system to extract data from all of them.

You do not need to do this for the first release of the Data Warehouse. Using the top-down approach, some data sources can be left for later.

Consider the case of the DW being built to consolidate corporate reporting, particularly after an acquisition or merger. By definition, the task is to gather all the sources of financial data from all the far flung corners of the corporation. This is enough of a challenge, but there are still data sources that can be left for later such as manufacturing performance measurements, sales and marketing plans, human resources etc.

Data Consistency and Harmonizing

As we saw above, the disparate data elements we have identified for the Data Warehouse have a surprising degree of inconsistency. This is expected if the company has acquired subsidiaries over time but can be true even within long standing departments.

The usual approach to solving this problem is to transform the data before loading it to the warehouse and make it all consistent. (See 'Extract, Transform and Load' below).

There is a better way and that is not to transform but to *harmonize* the data at a later stage. As raw data comes from the source systems in the *extract* phase, load it direct to tables. Then, build relationship tables that harmonize the different key values. (Example: different manufacturing subsidiaries use different item numbers with different formats for the same item. Leave them in their raw form as item data loads to the warehouse and build a table with the relationships between them, a common description for end users and a 'primary' value of the item number).

The reason for keeping data in its raw form is for validation. We have to have a way to verify the accuracy of every number in the data warehouse. Users will often query a value, especially if it is not the expected one and even more so if it brings bad news. By keeping data in its raw form, we can always capture the original data to present to a questioning user (or in these days of Sarbanes-Oxley, an auditor). If we are dealing with intermediate transformations of the data between its raw form and what we store in the warehouse, this process is made much more difficult. We have to justify not only the data but the algorithm transforming it as well.

The purpose of the 'common description' and 'primary value' mentioned above is for end user reporting and for aggregation.

A report showing production rates of widgets at different plants must show a single description for the widget and a common item number. Users don't want to do the harmonizing themselves, that is what the warehouse is for.

As we aggregate the data (items roll up into product lines or brands for example), a single 'primary' value for the item number is essential. Aggregation requires another relationship table defining the levels of data. For example, the items that make up a product line; the product lines that make up a business unit. (See below).

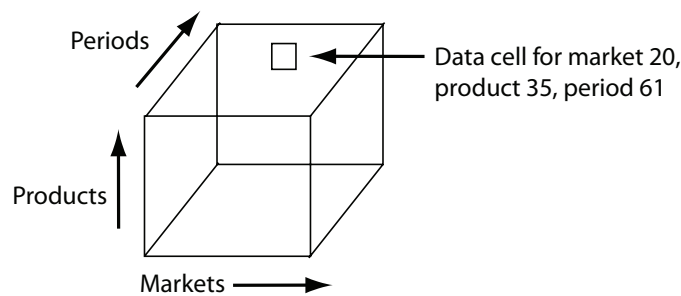
Aggregation is where the company's business rules get applied. Here we have to deal not only with the items that make up a product line but differences in the quantitative aggregation. For example, "quantity on hand" is a straightforward number, easily defined in a business rule. "Cost allocation", defining how manufacturing costs are passed to the corporate level is probably a much more complex business rule.

Understanding Dimensional and Relational Data Models

Data warehouses use either the relational or dimensional models. Each model has benefits and drawbacks.

Dimensional model

A Dimensional model stores data in a data cube or multi-dimensional matrix. In our example on the following page, this would be a three-dimensional matrix with markets along one side, products and periods along the other two, and the cells containing the data facts for each market/product/period intersection. (The measures dimension disappears into the cube itself)



This matrix structure allows access to any market/product/period cell directly. It can extract a slice or a block of data quickly by rotating its axes and re-ordering data easily. This is useful for statistical analysis (cross tabulating and correlation) and for output formatting.

For these reasons, almost all Business Intelligence (BI) products employ a cube or matrix structure for their internal working data storage. However, the dimensional matrix structure has severely limited data storage capabilities when you consider that most real data bases have a low density factor.

Density Factor

A typical mid-sized beverage manufacturing company has thousands of supermarket chains and distributors they sell to, probably thousands of individual product items (UPCs or SKUs), and wants to keep weekly back data in their warehouse database for two years.

The possible intersections of customer, product, and week are enormous. Consider this example:

$$\begin{array}{r}
 5,000 \text{ customers} \\
 \times 5,000 \text{ products} \\
 \times 104 \text{ weeks} \\
 \hline
 = 2,600,000,000 \text{ possible sales records.}
 \end{array}$$

Of course, not all customers order all products (unfortunately) or do not place orders every week. New products arrive and others are phased out reducing the active SKU count. Products are often sold regionally, again reducing the customer – to – product overlap.

The actual number of customer/product/week intersections is completely dependent on the company and its business. This density factor can be as low as one percent.

Even at one percent density factor, our example mid-sized company has 26 million rows in the data (fact) table. This is not a trivial number. Today, all large scale Data Base Management Systems (DBMS) can handle this comfortably. However, consider the load on their data server when 100 data analysts want to access it simultaneously and want their data queries answered instantly. Data warehouses require significant computer power if they are going to work properly for the benefit of their users.

To gain all the benefits of the matrix data model, the matrix must be fully populated. As we saw, our mid-sized company would have a matrix containing 5000 x 5000 x 104 or 2.6 billion (5000 x 5000 x 104) cells but a density factor of 1%. What about these non-existing data intersections? Dimensional products employ a compression technique to save the unused space from non-existing data intersections.

This compression process degrades access performance. It also makes it difficult to populate new cells in the interior of the matrix. (Some dimensional DBMS products require a complete rebuild to do this). Dimensional models work very well with small scale databases but have severe limitations with medium and large data bases, (the very nature of data warehouses!).

Relational model

The relational model uses a straightforward database table to store fact information. With the relational model the non-existing data intersections simply do not exist in the data table. This minimizes data storage and permits much larger databases. New intersections are appended to the end of the table.

Query retrieval times for small databases are comparable between relational and dimensional products, but far, far superior with the relational model for large databases.

Generally, the **relational model is preferred for data warehousing** because:

- it is easy to update
- it can support very large databases
- all but the smallest companies already own a relational DBMS product with the corresponding skilled engineers
- dimensional model products are proprietary

According to the Winter Corp. annual survey of very large databases used for Business Intelligence (BI) work, the ten largest all use a relational DBMS.

Understanding Data Warehousing and the Relational Data Model

It is important to understand how data is stored and used for business intelligence. There are many variables to data warehousing: the data schema, data density, aggregations, the characteristics, and the size of the data warehouse. The following considerations must be taken into account when designing a data model that provides the best performance and flexibility.

Consider the design of a data warehouse for our beverage company. We are using the top-down approach and the marketing department is our first users.

Data Models for a Marketing Research Data Warehouse

In a relational database, information is stored in tables with columns and rows. One table may contain columns of descriptive information; another table may contain columns of numeric data.

Simple Single Table

The simplest marketing research database would appear as in the following table, which contains weekly sales and price data.

The database below has two dimensions. The period dimension has one element, Week, with shows the values of Week ending Jan 7 2006, Week ending Jan 14 2006, and Week ending Jan 21 2006 etc... The fact dimension shows the values of three elements: Dollar Sales, Unit Sales, and Price.

Week	Dollar sales	Unit sales	Selling Price
Week ending Jan 7 2006	10	5	2.00
Week ending Jan 14 2006	12	6	2.00
Week ending Jan 21 2006	14	7	2.00

Four Dimensional Single Table

This is far too simple: we need to know which products are selling and where. The database will have several dimensions in a single table. The following example contains market, product, period, and fact dimensions.

Market	Product	Week	Dollar sales	Unit sales	Price
CENTRAL	SNAP SODA	Week ending Jan 7 2006	10	5	2.00
CENTRAL	SNAP SODA	Week ending Jan 14 2006	12	6	2.00
CENTRAL	SNAP SODA	Week ending Jan 21 2006	14	7	2.00
CENTRAL	CRACKLE POP	Week ending Jan 7 2006	9	3	3.00
CENTRAL	CRACKLE POP	Week ending Jan 14 2006	12	6	2.00
CENTRAL	CRACKLE POP	Week ending Jan 21 2006	16	8	2.00
EAST	SNAP SODA	Week ending Jan 7 2006	10	4	2.50
EAST	SNAP SODA	Week ending Jan 14 2006	15	6	2.50
EAST	SNAP SODA	Week ending Jan 21 2006	15	6	2.50
EAST	CRACKLE POP	Week ending Jan 7 2006	10	4	2.50
EAST	CRACKLE POP	Week ending Jan 14 2006	15	6	2.50
EAST	CRACKLE POP	Week ending Jan 21 2006	20	8	2.50

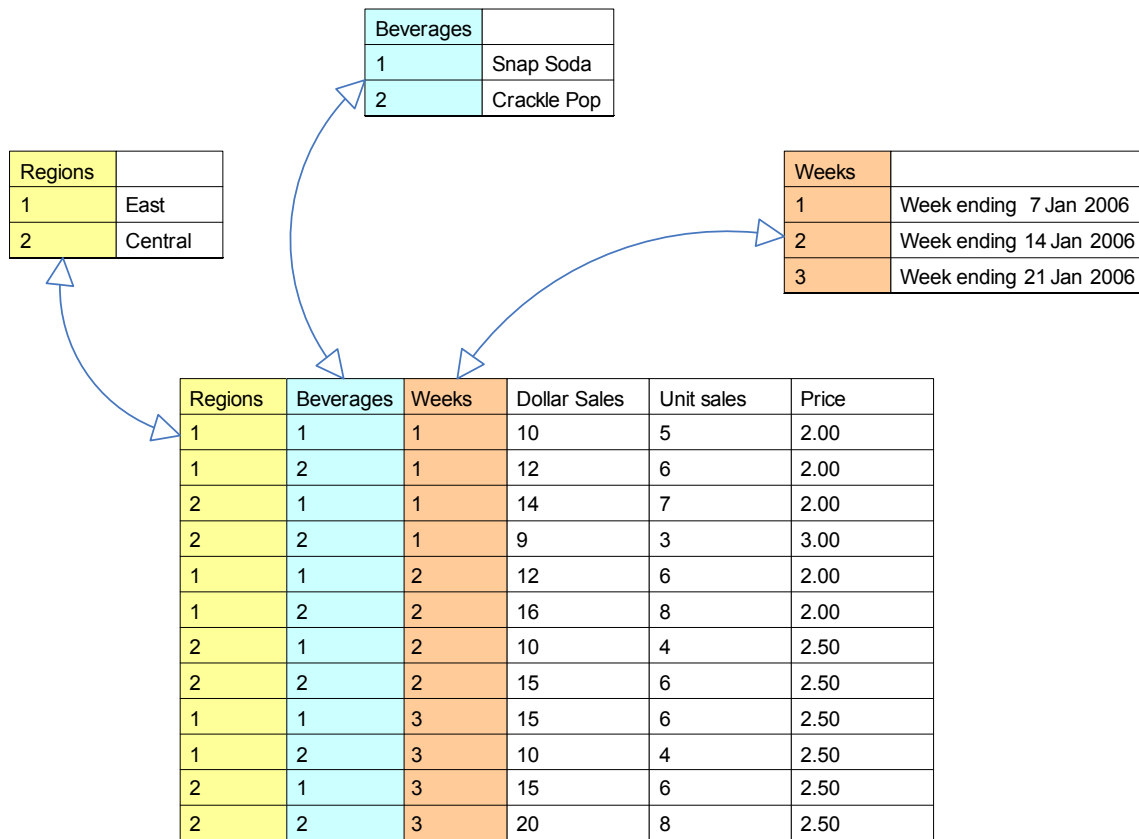
Star Schema

In reality, no data warehouse is as simple as these examples. As we saw above, our company has 5000 customers, 5000 products and 104 weeks of data. Our density factor of 1% produces 2,600,000 rows. The first issue to resolve is that repeating the names of the markets and products throughout this table is too inefficient in data storage.

To solve this problem, data warehouses are usually organized into a star schema. To make the above example into a star schema, we would replace the descriptions for markets, products, and weeks with the smallest numeric value that can identify each. Then we would add a reference table that links the numeric value to the description.

A star schema has one data (or fact) table that contains a key column for each dimension. Each key column is associated to its reference table, which contains the descriptions for the dimension. Then the fact table has columns for each measure.

Figure 1: Star Schema



Each of the dimensions is arranged in a star form around the fact table.

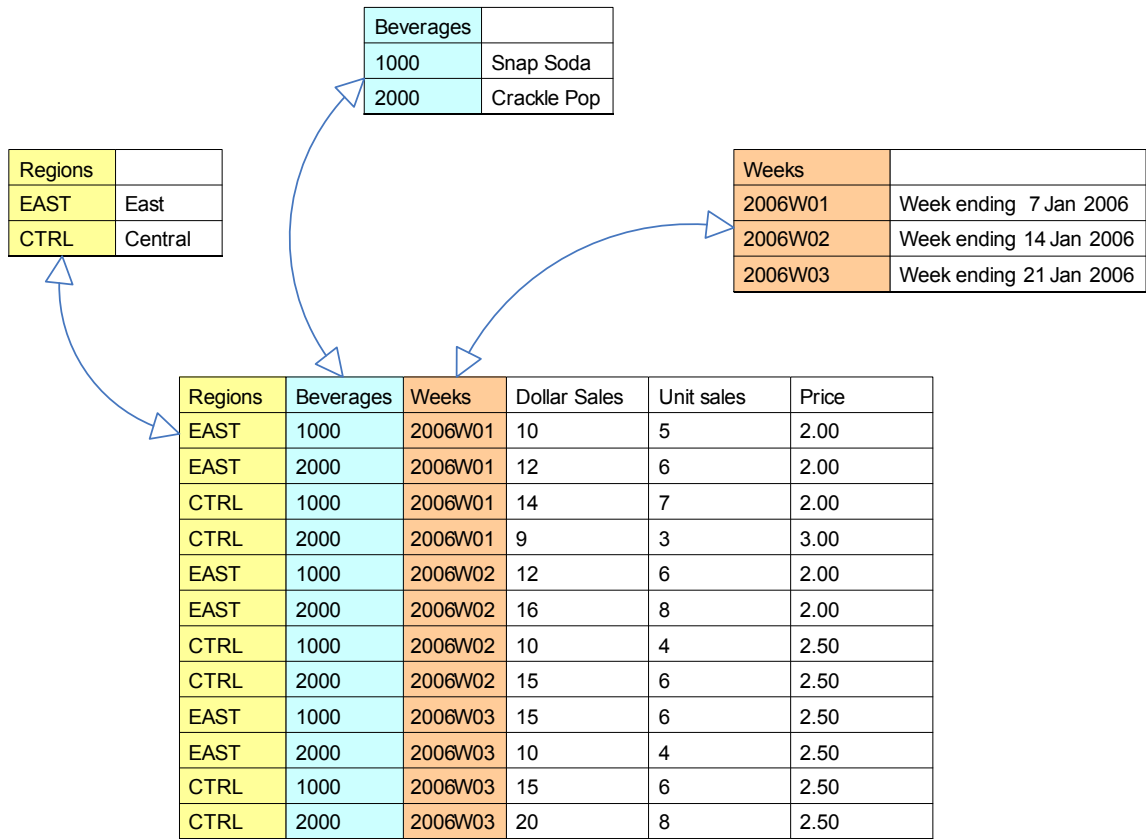
We have saved a lot of space by using small number values instead of the full descriptions. Data Modelers call these keys "surrogate keys"; purists in the data base industry maintain that all data warehouses should use them. They minimize space and maximize performance (historically, database systems prefer numeric keys to alphabetic keys).

There is a problem however with surrogate keys, the data in the fact table is meaningless without reference to the dimension or reference tables. All companies have short alphanumeric abbreviations for their geographic entities and products. These exist in their transaction processing systems and can easily be imported into the data warehouse.

Our opinion at Ironbridge Software is that meaningful "business keys" are far preferable to numeric surrogate keys. The days when Data Base Management Systems did not perform well with alphanumeric keys are long over. There is one further problem with surrogate keys; someone has to make them up, make sure they are unique, maintain the list and make sure no keys ever get re-used. We believe that this is simply unnecessary work for no benefit.

The 'business key' schema looks like this.....

Figure 2: Star Schema with Business Keys



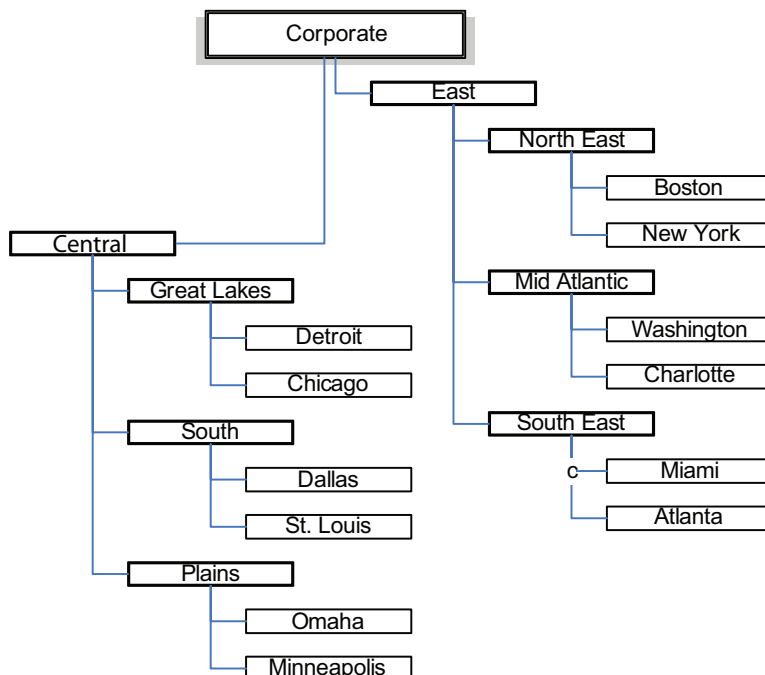
Note that we chose to use the internal SKU number for "Snap Soda" and "Crackle Pop"

Reference (or dimension) Tables

In our geography dimension, we have just 2 regions, EAST and CENTRAL. Next, we want to add the districts below region and then sales offices below districts and finally the customers handled by each sales office.

Figure 3: Geography Hierarchy.

(We will omit showing the 5000 customers to save trees.)



How do we represent this data in a relational database table? There are 3 options.

Option 1: A parent-child style table

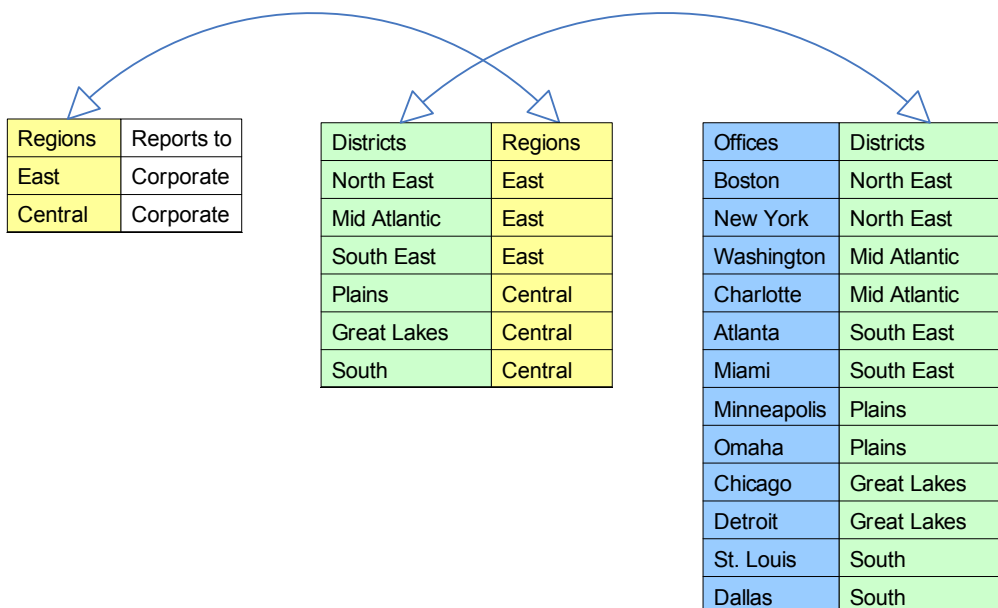
Geography	Reports To
Corporate	
East	Corporate
North East	East
Mid Atlantic	East
South East	East
Central	Corporate
Plains	Central
Great Lakes	Central
South	Central
Boston	North East
New York	North East
Washington	Mid Atlantic
Charlotte	Mid Atlantic
Atlanta	South East
Miami	South East
Minneapolis	Plains
Omaha	Plains
Chicago	Great Lakes
Detroit	Great Lakes
St. Louis	South
Dallas	South

This is the simplest way to represent the relationship between regions, districts and sales offices. Notice though that we have no idea just by looking at the data which entities are regions or districts or offices. Another problem with this style is that to display this data in a meaningful way to an end user is difficult with the SQL language. In technical terms, it requires a series of equi-joins for each level in the hierarchy to retrieve one path.

(Data Base Management Systems use a standard language called SQL – Structured Query Language, developed by IBM in the late 1970's.)

Option 2: Linked tables

Figure 4. Linked Geography Tables



These tables are easily understandable and they require only a little more space than the 'Parent-Child' style table. They still present 2 significant problems.

1. The SQL required to present an organization chart to the user is complex. Technically, each table must be joined to its neighbor to traverse the organization tree. With our 3 level organization chart, this is simple, but with more levels this quickly becomes complex and slow.
2. This model does not support 'National Accounts'; customers who are serviced from more than one sales office. In fact, generally, it does not support business analysis queries that go across sales offices. For example, the shipping department wants to divide customers into those we service through their distribution centers and those we service through direct store delivery without having to sort through every sales office. Marketers want to divide customers by how we sell to them (through direct sales or through brokers) to know how to target their expenditures, again, without having to sort through each sales office.

(When we add these link tables to the star schema, they become a 'snowflake' schema.)

Option 3: Embedded characteristics table

Figure 5 Embedded Characteristics Table

Region	District	Office	Name	Level	Delivery	Marketing
East			East	Region		
East	North East		North East	District		
East	North East	Boston	Boston	Office		
East	North East	New York	New York	Office		
East	Mid Atlantic		Mid Atlantic	District		
East	Mid Atlantic	Washington	Washington	Office		
East	Mid Atlantic	Charlotte	Charlotte	Office		
East	South East		South East	District		
East	South East	Atlanta	Atlanta	Office		
East	South East	Miami	Miami	Office		
Central				Region		
Central	Plains		Plains	District		
Central	Plains	Minneapolis	Minneapolis	Office		
Central	Plains	Omaha	Omaha	Office		
Central	Great Lakes		Great Lakes	District		
Central	Great Lakes	Chicago	Chicago	Office		
Central	Great Lakes	Detroit	Detroit	Office		
Central	South		South	District		
Central	South	St. Louis	St. Louis	Office		
Central	South	Dallas	Dallas	Office		
East	North East	Boston	Stop In	Customer	DSD	Direct
East	North East	Boston	Shores	Customer	DC	Direct
East	North East	Boston	B&Q	Customer	DSD	Broker
East	North East	New York	B&Q	Customer	DSD	Broker
East	North East	New York	Goliath Stores	Customer	DC	Broker
East	North East	New York	David Stores	Customer	DC	Direct

This style is the most effective for Data Warehousing. It requires more space than the others but provides the best performance and flexibility.

All relationships in the geography dimension are in this table. The introduction of a 'level' column is key to the flexibility. This determines the relationships on any given row of the table and makes the SQL much simpler and avoids joins between tables.

Customers exist at the 'Customer' level and have additional characteristics beyond region, district and office. These characteristics are the key to data mining; the ability to 'slice and dice' the data and to make non-hierarchical, or 'orthogonal' relationships.

(Note that for large dimensions, we may still want to replace the descriptions in this table with short abbreviations and use look up tables for full descriptions).

Characteristics and Attributes

The geography dimension in our example has characteristics: 'region', 'district', 'office', 'delivery' and 'marketing'.

The terms characteristics and attributes are only loosely defined in data warehousing and are often used interchangeably. At Ironbridge, we use the term attribute to indicate a purely hierarchical indicator such as 'region', 'district' and 'office'. We use the term characteristic to indicate a cross-hierarchy relationship such as 'delivery' and 'marketing'.

Product dimension

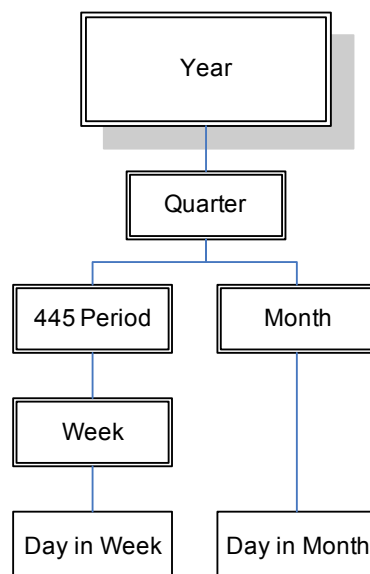
Everything we have discussed for the geography dimension applies equally to products. Product attributes (hierarchical) are 'category' and 'brand'. Characteristics go across brands and possibly categories, for example, 'size', 'packaging', 'segment'. The lowest level of the product dimension is item, either SKU or UPC. The lowest level of a dimension is called the atomic level.

There is often a significant difference between a stock keeping unit (SKU) and the universal product code (UPC) that applies to an item. For example, a product that is sold in both a glass bottle and a plastic bottle will have 2 SKU's and possibly be produced in separate plants. To the consumer, the product inside is identical and priced the same, so it has one UPC.

The opposite is true when a product is sold on promotion: the label says '30cents off!' and it must have a different UPC from the regular priced item to identify it as it crosses the supermarket scanner. To manufacturing and shipping, it is the same product and has one SKU.

Time Dimension

Time is the simplest dimension to build and manage, but it still requires a dimension table similar to geographies and products. There is a natural hierarchy or possibly 2 as below:



Most companies operate on either a 4-4-5 periods or a calendar month basis, but the data warehouse may have to support both if subsidiaries operate differently, or (more likely) customers operate differently. Retailers use a February to January calendar for example. When our users want to make presentations to retailers using the data warehouse, they want to use the retailer's time alignment. Internally, of course, users want to use our own alignment.

This situation of parallel alignments applies also to the geography and product dimensions. When our users present data to a customer, they want to use the customer's regional (geographic) alignment, likewise to show the customer's product SKU's, not ours.

Fact (Measures) Dimension

The terms 'measures' and 'facts' are used in data warehousing interchangeably. Data Modelers refer usually to the 'fact dimension' which can contain either 'facts' or 'measures'.

The fact dimension exists in a relational database as one (or more) tables. It has one column for each fact such as 'Dollar Sales', 'Sales Units', or 'Price' and one column for each dimension key, that is a geography key, product key and time key.

This is the fact table for our example beverage company.....

Regions	Beverages	Weeks	Dollar Sales	Unit sales	Price
EAST	1000	2006W01	10	5	2.00
EAST	2000	2006W01	12	6	2.00
CTRL	1000	2006W01	14	7	2.00
CTRL	2000	2006W01	9	3	3.00
EAST	1000	2006W02	12	6	2.00
EAST	2000	2006W02	16	8	2.00
CTRL	1000	2006W02	10	4	2.50
CTRL	2000	2006W02	15	6	2.50
EAST	1000	2006W03	15	6	2.50
EAST	2000	2006W03	10	4	2.50
CTRL	1000	2006W03	15	6	2.50
CTRL	2000	2006W03	20	8	2.50

Notice that a fact table will always use short abbreviations, not long descriptions for the keys to save space.

The example shows a high level aggregate of data, we are looking at regional sales for our two product lines at a weekly total level.

This data comes from aggregating a much lower, more detailed level. Data Warehouses get their data from the company transaction systems where (by definition) they record one customer buying one product at a point in time. At this lowest level, one row in the fact table represents one business transaction. The keys uniquely identify the customer, item purchased and the date.

Low level tables have low density factors, higher levels such as in the example have 100% density factor, that is, every product sells in every region in every week.

Relational databases save space by omitting data that does not exist. If a product line does not sell in a region in a week, there would simply be no row for that intersection of keys.

Note though that there is a difference between non-existent data and zero data. If a product does not sell because it is not marketed in a region, or has gone obsolete, then there is no entry and users see it as 'Not Available'. If the product is marketed but there have been zero sales, then there should be a record for this.

Our table contains a column 'Price'. This is a simple calculation of 'Dollar Sales / Unit Sales'. We don't need to store a simple calculation like this and usually it would not appear, we would calculate it on demand. This calculates 'Average Selling Price'. If 'Price' on the other hand is 'recommended retail price' or 'list price' then we would store it since it cannot be calculated from the raw data.

Data Marts

Data Marts are small data warehouses. They are of two varieties.

1. Small data sets gathered for a specific purpose such as budgeting, manufacturing productivity analysis etc.
2. Extracts from the larger data warehouse for analysis with a particular software tool. This often happens when a dimensional (cube) based tool has specific analytic capabilities or the full data warehouse is too large for the product.

Some writers consider a collection of Data Marts to constitute a Data Warehouse and even that such a collection is a valid data model for a data warehouse. At Ironbridge, we consider this to be true only when the separate marts can be accessed as a consistent whole.

To make a disparate set of data marts appear consistent requires harmonizing them (discussed later). The effort to do this is significant. If the data marts exist in non-compatible software systems then the effort is usually cost prohibitive. The most common approach is to copy the data marts into the warehouse, harmonizing the data in the process.

There is a notable exception to this approach. In the field of marketing research, the databases owned by the providers of supermarket scanner data are among the largest in the world. Integrating them into corporate data warehouse is impossible with today's technology. Ironbridge's Net-Bench product can make these databases appear as part of a data warehouse by using real-time integration and harmonizing techniques.



IRONBRIDGE
SOFTWARE

ATLANTA

2390 Hopewell Plantation Drive
Alpharetta, GA 30004
678-333-3017

CHICAGO

2700 S River Road
Suite 120
Des Plaines, IL 60068
847-699-3369

NEW YORK

16 Sullivan Drive
Basking Ridge, NJ 07920
908-470-2995