Praise for "Super Charge Your Data Warehouse"
and
The Data Vault Model and Methodology

Most Important Design Technique

"Data Vault represents the most important and most basic design technique for integrity of data and audit-ability of data in the data warehouse environment. Built on a DW 2.0 foundation, Data Vault advances the state of the art for data base design for those organizations where auditability of data is an issue.

In addition data vault allows the organization to manage data whose semantics and structure change over time. Classical data base design approaches have required a redefinition and a reorganization of data every time there is a need for a structural change in data.

But with data vault you can change your data structures gracefully over time, thus saving data base administration time and machine resources required for data unload/reload …"

Bill Inmon, Father Of Data Warehousing

Truly Groundbreaking Innovation

"Dan has devised one of the only truly groundbreaking innovations in information architecture over the past twenty years

The Data Vault should be considered as a potential standard for RDBMS-based analytic data management by organizations looking to achieve a high degree of flexibility, performance and openness …"

Doug Laney, Deloitte Analytics Institute

"This enables organizations to take control of their data warehousing destiny, supporting better and more relevant data warehouses in less time than before."

Howard Dresner, Dresner Advisory Services

I Keep A Copy On My iPad

"This book captures a practical body of knowledge for data warehouse development which both agile and traditional practitioners will benefit from. I keep a copy on my iPad so that I have ready access to it when working with clients. 'Nuff said…."

Scott Ambler, Author Of Agile Modeling,
Agile Database Technologies and Several Other Books

"The Data Vault is a foundationally strong and an exceptionally scalable architecture …"

Stephen Brobst, CTO, Teradata

A strong and enduring foundation

"Data Vault Modeling is a strong basis for the majority of the data warehouse implementations. The conceptual simplicity of the data structure makes it easy to populate, extract, extend and explain…"

Mark Zwijsen, The Netherlands

"Tearing down and building up … By tearing down your data stream into hubs, links and satellites you gain a lot of flexibility:

- Flexibility to recreate you source data to a certain point in time. (Great for compliance)

- Flexibility to easily adapt to ever changing reporting requirements.

- Flexibility to automate the data warehousing process and thus save a lot of time and money.

Great reasons to adopt Data Vault Modeling and Methodology! …"

Marco Schreuder, Certified Data Vault Modeler

Highly Scalable Modeling Method

"Relational and dimensional-based integration layers have served the business intelligence discipline well…until now. The future is the Data Vault, an innovative, hybrid approach that draws upon the strengths of relational, dimensional, and highly-normalized modeling architectures resulting in a fresh, process-based, rule-guided, and highly scalable modeling method. There is no going back now …"

Randy Benzel

It Is A Must Read

"Data Vault modeling combines the best of 3NF and Dimensional modeling to allow building fast, flexible, reliable and modular Data Warehouses.

This book answers the what, how & why questions of Data Warehousing.

It is a must read for everyone building or planning to build a Data Warehouse. …"

Gabor Gollnhofer

Flexibility and Simplicity

"I've been dealing with issues related to data integration and constructing physical models of databases many years and I am absolutely convinced that if the architecture and data model is designed correctly, then most problems can be avoided.

Familiarity with the methodology of data Vault allowed me to see old problems in new light.

I found two reasons to adopt Data Vault Methodology

First – flexibility and simplicity modelling data structure; Second – technical issues such as: building more optimal storage of the date, parallel data overloading and partitioning."

Dmitry Pushkashu

Foreword by Howard Dresner

Data warehousing has long been a discipline shrouded in complexity, misunderstanding and consternation. However, like the construction of a building, which may collapse without a solid foundation, a data warehouse is fundamental to the viability of any Business Intelligence solution. And, if done properly, a data warehouse should serve as an important business differentiator – enabling users to more quickly discern critical events and trends which directly impact the success of the organization.

Due to inexperience and trial and error, many organizations have endeavored to build their data warehouse only to learn painful lessons. They overspend budgets, have little to show after extended periods of time and alienate users who want something quickly and lack an understanding of the difficulty and complexity surrounding the preparation of the underlying data. In fact, there are organizations that completely eschew data warehousing due to this taint of project failure. Yet, without a well-conceived, architected and maintained data warehouse, the Business Intelligence solution will inevitably fail.

These issues have been exacerbated by several factors. With the advent of ERP systems, pre-packaged data marts and warehouses have come to the fore, offering functionally and vertically specific offerings – with the promise of rapid "time to action". Although certainly appealing, customers are confronted with either accepting the vendor's "standard" view of the business through preconceived metadata, schemata and reports or face substantial systems integration costs to create and maintain a customized environment. The other factor is an increased focus upon end user environments and tools with the disintermediation of IT-based solutions in favor of vendor solutions which cater to users. These solutions may ignore data warehousing entirely as it rarely helps to sell end user software tools. As a result, end users have virtually no appreciation for the need or complexity surrounding data warehousing.

The only way to counter these phenomena and get organizations refocused on Business Intelligence success is through a two-pronged approach. First, organizations need to take advantage of decades of knowledge and know-how surrounding the conception and design of data warehousing. Second, a meaningful dialogue needs to be established between those responsible for the data warehouse and the end user consumers.

Dan's latest book (and his others as well) can help on both counts. First, he provides data warehousing professionals with useful methods and process tools for the construction of more well-conceived, flexible and extensible data warehouse architecture. This enables organizations to take control of their data warehousing destiny, supporting better and more relevant data warehouses in less time than before. But, perhaps even more importantly, Dan's work is straightforward and highly understandable. When read by end users, it can form the basis of a common understanding, vocabulary and dialogue with data warehouse professionals.

I applaud Dan's contribution to the body of Business Intelligence and Data Warehousing knowledge and recommend this book be read by both data professionals and end users – who have a common stake in the success of the organization and the information needed to support it.

Howard Dresner

Author of Profiles in Performance, Business Intelligence Journeys and the Roadmap for Change (2009 – John Wiley & Sons) and The Performance Management Revolution, Business Results Through Insight and Action (2007 – John Wiley & Sons)

Chapter 1
INTRODUCTION AND TERMINOLOGY

Welcome to the technical book about Data Vault Modeling. This book is for practitioners and implementers. The content of this book is focused on the Data Vault structures, definitions of the structures, metadata, and data modeling – it does not cover the loading, querying, nor partitioning of the Data Vault. These feature sets will be covered in the next set of technical books.

1.1 Do I need to be a Data Modeler to read this book?

No, it is not necessary to be a data modeler to read this book. While a data modeling background is helpful, it is not required. The writing covers the basic components of the Data Vault Model, and also introduces information about the concepts utilized by nearly all relational database systems. Experience with RDBMS engines also can be applied to the concepts and knowledge presented here. This book also assumes you are familiar with the basics of data warehousing as defined by W.H. Inmon and Dr. Ralph Kimball.

A common understanding of fields / columns, tables, and key structures (such as referential integrity) is helpful. In the next section are descriptions of common terms used throughout this book.

1.2 Review of Basic Terminology

The terminology in this book consists of basic entity-relationship (E-R) diagramming and data modeling terms. Terminology such as Table, Entity, Attribute, Column, Field, Primary Key, Foreign Key, and Unique Index are utilized throughout. For reference purposes the following basic level definitions of the terms are provided.

Term	Definition
Table	A composite grouping of data elements instantiated in a database, making up a concept.
Entity	A table, as referred to in a logical format (eg: customer, account, etc.)

Term	Definition
Attribute	A single data element comprised of a name, data type, length, precision, null flag, and possibly a default value.
Column	An ordered attribute within a table.
Field	Same as Column. See Column definition.
Primary Key	Main set of one or more attributes indicating a unique method for identifying data stored within a table.
Foreign Key	One or more attributes associated with the primary key in another table. Often used as lookup values, may be optional (nullable) or mandatory (non-null). When enabled in a database, foreign keys insure referential integrity.
Unique Index	One or more attributes combined to form a single unique list of data spanning all rows within a single table.
Business Key	Component used by the business users, business processes, or operational code to access, identify, and associate information within a business operational life-cycle. This key may be represented by one or more attributes.
Natural Key	See business key.
Relationship	An association between or across exactly two tables.
Many to 1	A notation used to describe the number of records in the left-hand table as related to the number of records in the right-hand table. Example: many customer records may have 1 and only 1 contact record.
Many to Many	An open-ended notation. For example: where many customer records may have many contact records.
1 to 1	A notation dictating singular cardinality: 1 customer record may have 1 and only 1 contact record.
Cardinality	In mathematics, the cardinality of a set is a measure of the "number of elements of the set". For example, the set A = {1, 2, 3} contains 3 elements, and therefore A has a cardinality of 3. There are two approaches to cardinality – one which compares sets directly using bijections and injections, and another which uses cardinal numbers. Reference: http://en.wikipedia.org/wiki/Cardinality
Constraint	A relationship between two tables that enforce existence of data in a parent table, or an indicator of uniqueness and not-null column. A constraint may also indicate basic rules such as defaults, or functions to check values, or possibly ranges of data.
Weak Relationship	A constraint that is optional (when the data is not null, then the constraint is checked for validity). When the data is null, the constraint is not checked for validity.

Term	Definition
Strong Relationship	A constraint that is non-optional. Data is required (not-null) in the child table all the time, and therefore is checked for validity.
Associative Entity	An associative entity is an element of the Entity-relationship model. The database relational model doesn't offer direct support to many-to-many relationships, even though such relationships happen frequently in normal usage. The solution to this problem is the creation of another table to hold the necessary information for this relationship. This new table is called an associative entity. http://en.wikipedia.org/wiki/Associative_entity

Data models are diagrammatic representations of information and classes of information to be held within a mechanical storage mechanism such as a database engine; except in the case of a conceptual/business model which should be independent of technology. Common database engines today include: DB2 UDB, Teradata, MySQL, PostGreSQL, Oracle, SQLServer, and Sybase ASE. There are several main notations used for E-R diagrams (e.g., Chen, Barker, IDEF, etc). An example of an E-R diagram using Elmasri/Navathe notation is below:

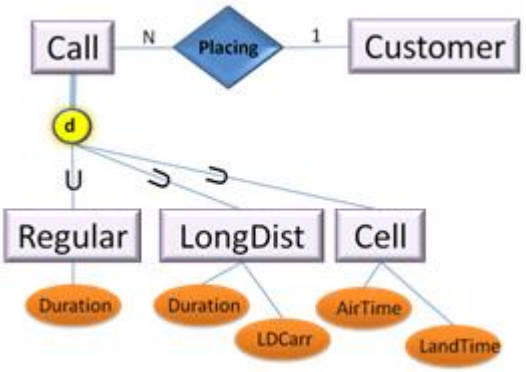


Figure 1-1: Example E-R Diagram (Elmasri/Navathe)

Image: http://cisnet.baruch.cuny.edu/holowczak/classes/9440/entityrelationship/

Data models (such as E-R diagrams) house linguistic representations of concepts tied together through associations. These associations can also be thought of as Ontologies. There are many types of data modeling notations available in the world today. Two main types are focused on in this document: 3rd normal form and Star Schema. For reference purposes, simple definitions of both styles are included below.

3rd Normal Form is defined as follows:

> The third normal form (3NF) is a normal form used in database normalization. 3NF was originally defined by E.F. Codd[1] in 1971. Codd's definition states that a table is in 3NF if and only if both of the following conditions hold:
>
> - The relation R (table) is in second normal form (2NF)
> - Every non-prime attribute of R is non-transitively dependent (i.e. directly dependent) on every key of R.

A non-prime attribute of R is an attribute that does not belong to any candidate key of R.[2] A transitive dependency is a functional dependency in which X → Z (X determines Z) indirectly, by virtue of X → Y and Y→Z (where it is not the case that Y → X).[3]

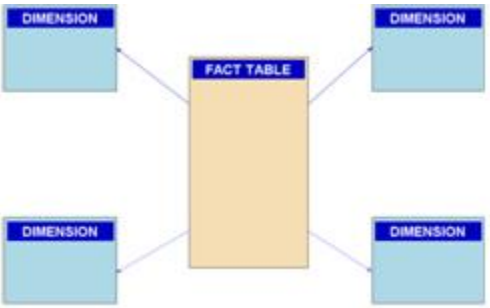
A 3NF definition that is equivalent to Codd's, but expressed differently was given by Carlo Zaniolo in 1982. This definition states that a table is in 3NF if and only if, for each of its functional dependencies X → A, at least one of the following conditions holds:

- X contains A (that is, X → A is trivial functional dependency), or
- X is a super key, or
- A is a prime attribute (i.e., A is contained within a candidate key)[4]

Zaniolo's definition gives a clear sense of the difference between 3NF and the more stringent Boyce-Codd normal form (BCNF). BCNF simply eliminates the third alternative ("A is a prime attribute").
http://en.wikipedia.org/wiki/Third_normal_form

Star schema modeling is defined as follows:

The star schema (sometimes referenced as star join schema) is the simplest style of data warehouse schema. The star schema consists of a few "fact tables" (possibly only one, justifying the name) referencing any number of "dimension tables". The star schema is considered an important special case of the snowflake schema. http://en.wikipedia.org/wiki/Star_schema



The Star Schema modeling approach is "called" a star, because it appears to look similar to a star-like shape. Star Schema modeling is championed by Dr. Ralph Kimball.

1.3 Data Modeling Notations Used in This Text

Crows Foot notation is utilized throughout this text to represent raw data models; in addition to the crows-foot notation this text introduces arrows to represent data migration paths (vectors/direction of data flow). It is occasionally easier to describe the vector notation to business users when compared with describing crows-foot notation.

"The "Crow's Foot" notation represents relationships with connecting lines between entities, and pairs of symbols at the ends of those lines to represent the cardinality of the relationship. Crow's Foot notation is used in Barker's Notation and in methodologies such as SSADM and Information Engineering." *http://en.wikipedia.org/wiki/Entity-relationship_model*

Figure 1-2: Crows Foot and Arrow Notation Example

Arrow notation is less descriptive (mathematically) and shows only the direction or flow of the parent table primary key (e.g. Artist) into the child table (e.g. Song).

1.4 Data Models as Ontology's

Data models function as ontologies in this world. They seek to organize a hierarchy of information into a classification system. Ontologies are extremely powerful notions that can capture augmented or enhanced metadata (information about the data model) that is not represented by the model itself.

> "In both computer science and information science, an ontology is a formal representation of a set of concepts within a domain and the relationships between those concepts. It is used to reason about the properties of that domain, and may be used to define the domain.
>
> Ontologies are used in artificial intelligence, the Semantic Web, software engineering, biomedical informatics, library science, and information architecture as a form of knowledge representation about the world or some part of it." http://en.wikipedia.org/wiki/Ontology_(computer_science)

Ontologies are one way to represent terms beyond data modeling; much of the Data Vault model is based on the Ontology concepts. When the Data Vault model form is combined with the function of data mining, and structure mining then new relationships can be discovered, created and dropped over time. The ontology can be morphed or dynamically altered into new relationships going forward. There is more discussion on this topic in different sections of this book around the flexibility of the Data Vault model.

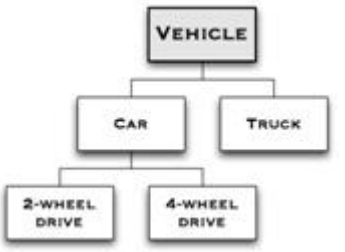


Figure 1-3: Small Example: Ontology for Vehicle

The ontology in Figure 1-3 is extremely simple and small. It represents the notion of the parent term vehicle which contains the sub-classes: Car and Truck. Car and Truck are both types of vehicles; however each has potentially different descriptors. Trucks generally contain larger frames, larger motors, larger wheels, and are capable of towing and hauling heavy loads where cars generally have a smaller turning radius, use less gas, and can house more people.

Ontologies are powerful categorization and organization techniques. Imagine a set of music on a mobile computing device. Now imagine that there are many different categorizations for that music, ranging from year, to composer, to album, to band, to artist, lead vocalist, etc… Now stack these categorizations in different orders or hierarchies – they function as indexes into the data set. At the end of the index are the same music files; they are simply categorized differently. This is the basic makeup of ontologies.

However, this description can go deeper; switch the existing categories out for business terms, and begin to describe each category. For instance: Genre. Different people might define

"what is classified as rock and roll" differently, but they are both right. Categorization is in the eye of the beholder, and is based on the individual's belief system and knowledge set (or context) surrounding the information at the bottom of the stack; which in this case are the music files.

The deeper into the ontology (or index), the more specialized and differentiated the definition becomes. For example: underneath Rock, there might be 70's, 80's and 90's rock, or there might be classic rock and roll. Where an individual who grew up in the 60's considers 60's and 70's to be a part of; while an individual who grew up in the 90's or later considers any music earlier than 1985 to be classic rock. This is just one of the issues that the Data Vault Model and implementation methodology provides a solution to. This book will uncover the key to modeling Ontologies in enterprise data warehouses for use with Business Intelligence systems.

In fact, learning warehousing, applying, and using ontologies is a critical success factor for handling, managing and applying unstructured data to a structured data warehouse. It is also a major component for operational data warehousing, along with business rule definition and dissemination of the data within an Enterprise Data Vault.

These are general descriptions of ontologies as used throughout this book. In addition to ontologies, data models typically contain short-hand notations for names of fields known as abbreviations. These abbreviations can have similar meaning within the same context (i.e. industry vertical) but may have different meaning across different context. For example: Abbreviation CONT in health care may mean contagious, in a legal system it may mean continuation. Abbreviations are best separated by vertical industry.

1.5 Data Model Naming Conventions and Abbreviations

Physical data models often contain abbreviations for classifying tables and fields as many RDBMS engines impose length limits on object names. The desire is to carry metadata meaning within the abbreviations which results in a data dictionary being created. The naming conventions usually start from the left hand side of the object name and move to the right with a logical flow with different parts of the abbreviations separated by an underscore. The typical abbreviation is made up of multiple components as shown in Figure 1-4:

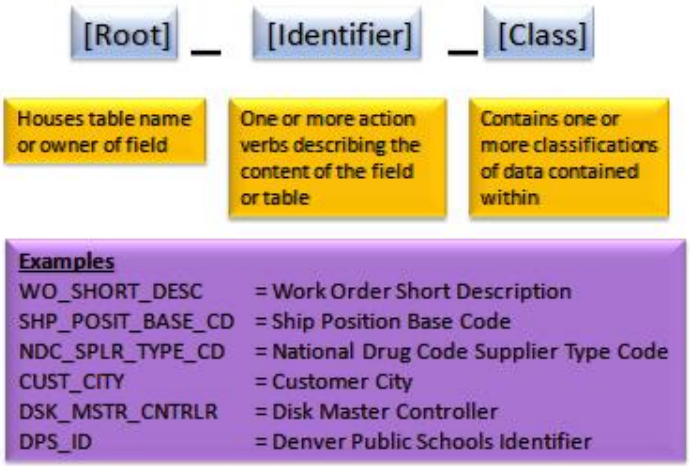


Figure 1-4: Example Abbreviations and Naming Conventions

NOTE: THESE NAMING CONVENTIONS ARE FOR THE PHYSICAL DATA MODEL. SOME OF THE EXAMPLES ON THE ENSUING PAGES ARE PHYSICAL, AND USE THIS APPROACH, WHILE OTHER EXAMPLES ARE LOGICAL AND DISPLAY BUSINESS TERMS. PLEASE TAKE NOTE OF THE DIFFERENCES BETWEEN THE LOGICAL AND PHYSICAL DATA MODELS.

Vowels may be kept in order to increase readability, however in general vowels are removed from the abbreviations for shortening reference names. There are some notations which do not include underscores rather they utilize text case to indicate the start of new terms (e.g., camel case used with SQL Server). The Data Vault physical modeling methodology recommends underscores as the best practice for abbreviations. The Data Vault logical modeling components recommends utilizing full business names which are demonstrated in this book.

NOTE: Naming conventions are one form of ontology and metadata that can be applied actively within the confines of the Business Intelligence world.

For the ontology listed in Figure 1-3 the abbreviations might be as follows:

- Vehicle = VEH
- Car = CAR
- Truck = TRK
- 2 Wheel Drive = TWOWHDRV, TWDRV
- 4 Wheel Drive = FOURWHDRV or AWD, FORWDRV

The suggested table naming conventions for the Data Vault are as follows:

- HUB , or H for HUB tables
- LNK, or L for standard Link Tables
- HLNK, or HL for hierarchical Links
- LSA, SAL, SLNK, SL for same as Links
- TLNK, TL for transactional Links
- SAT, or S for generic Satellites
- HSAT for Hub Satellites
- LSAT for Link Satellites
- PIT, or P for point-in-time tables
- BR, or B for Bridge tables
- REF, or R for reference tables

Within each of the Data Vault tables there are standardized fields (more on this later). The naming convention for these fields is as follows:

- LDTS, LDT for load date time stamps
- LEDTS, LEDT for load end date time stamps
- SQN for sequence numbers
- REC_SRC, RSRC for record sources
- LSD,LSDT for last seen dates
- SSQN for sub-sequencing identifiers

Always document the naming convention and the abbreviations chosen through a data dictionary in order to convey meaning to the business and the IT team. Naming conventions are vital to the success and measurement of the project. Naming conventions allow management, identification, and monitoring of the entire system – no matter how large it grows. Once the

naming convention is chosen, it must be adhered to (stick to it at all times). One way to insure this is to conduct frequent data model reviews and require non-conforming objects to be renamed

1.6 Introduction to Hubs, Links, and Satellites

The Data Vault model consists of three basic entity types: Hubs, Links, and Satellites (see Figure 1-5). The Hubs are comprised of unique lists of business keys. The Links are comprised of unique lists of associations (commonly referred to as transactions, or intersections of 2 or more business keys). The Satellites are comprised of descriptive data about the business key OR about the association. The flexibility of the Data Vault model is based in the normalization (of or separation of) data fields in to corresponding tables.

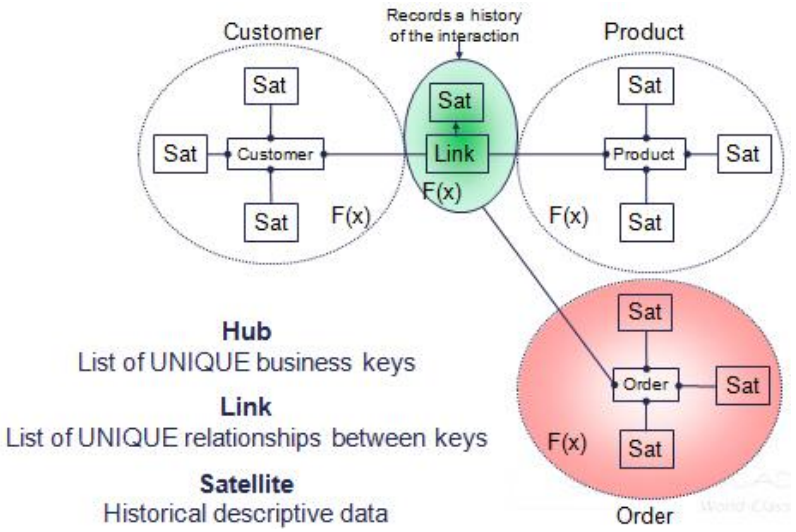


Figure 1-5: Example Data Vault

Data Vault models are representative of business processes and are tied to the business through the business keys. Business keys indicate how the businesses integrate, connect, and access information in their systems. Data Vault models are built based on the conceptual understanding of the business.

Concepts such as customer, product, order, email, sale, inventory, part, service, account, and portfolio are used to represent ideas that cross lines of business. Examples of lines of business may include: sales, finance, marketing, contracting, manufacturing, planning, production, and delivery. These concepts can be represented with business keys that should cross lines of business. The Hubs carry the unique list of these keys and are defined by semantic grain (granularity) and business utilization.

The Links represent association across the keys (in Figure 1-5 they show an Order actually being invoiced to a customer for a specific product). The associations change over time, some have

direction (akin to mathematical vectors), others are directionless. Links are physical representations of foreign keys, or in data modeling terms: an associative entity.

Hubs and Links do not contain context. Satellites provide the *context* defining the keys and associations *for a specific point in time.* The Satellites contain the descriptive data attributes about a Hub or Link that can change over time. Satellites are the *data warehousing* portion of the Data Vault model.

You see, Hubs and Links are like the skeleton and ligaments of the human body – without them we have no structure. Without them, our Data Warehouses are blobs of data loosely coupled with each other. But WITH them we have definition, structure, height, depth, and specific features. We as humans couldn't survive without a skeleton. The Data Warehouse cannot survive without Hubs and Links. They form the foundations of how we hook the data together.

For instance, the Hubs are like the different bones in the body, the arm, the leg, the toes, the head, etc… The Links are similar to the ligaments that hold the bones together, give them flexibility, and attach them in a specific order. Finally, the Satellites are added. Satellites are like the skin, muscle, and organs. They add color, hair, eyes, and all the other components we need to be described.

By separating the concepts of descriptive data from structural data, and structural data from Linkage data, we can easily begin to assemble a picture or an image of what our companies look like. The Hubs provide the working constructs to which everything is anchored. The Links provide the assembly structure to how these anchors interact, and the Satellites define the context (like hair color, eye color, skin type, etc.) of all of these components.

Remember this: the Data Vault is targeted to be an Enterprise Data Warehouse. Its job is to ***integrate*** disparate data from many different sources, and to ***Link*** it all together while ***maintaining*** source system context.

It sits between the source systems and the data marts. Why? Because the data marts are the ***interpretation*** layer of the integrated data warehouse data. In human terms think about it this way: think about a certain event that occurred in your life that you shared with another person. Do you both remember it the same way? Do you both remember the exact details? Or is your interpretation of the event slightly different than that of your friend?

Exactly my point, interpretation depends on context, and the context you use to remember the event is different than the context your friend uses. Even if the facts or the event itself is exactly the same (you were both there, you both saw the eclipse of the sun, but you both experienced it differently). This is why it's so important to separate ***interpretation*** from the ***facts.***

Let your Data Vault house the facts, and build your data marts to house the interpretation.

1.7 Flexibility of the Data Vault Model

The Data Vault model is built for extreme flexibility and extreme scalability. The Link table separates the relationships from the business key structures (the Hubs). The Link table provides

for the representation of the relationship to change over time. The Satellites provide the descriptive characteristics about the Hubs or Links as they change over time.

For instance, suppose you own a car and you are the registered driver. You currently have two relationships to the car: one as a driver, and one as an owner. Now suppose you hired a driver. Well, you still own the car right? Now, you have one relationship with the car as the owner, but the person you hired now has a relationship with the car as the driver. However, the description of the car has not changed.

What if you sold the car to someone else? Then your relationship with the car as an owner would END, and the buyer's relationship with the car would begin. This information about the relationship between business keys is what we keep in the Link structures. Again, the basic description of the car remains unchanged so the Satellite data is untouched.

The Link table may also be applied to information association discovery. Business changes frequently – redefining relationships and cardinality of relationships. The Data Vault model approach responds favorably because the designer can quickly change the Link tables with little to no impact to the surrounding data model and load routines.

MAJOR FUNDAMENTAL TENANT: THE DATA VAULT MODEL IS FLEXIBLE IN ITS CORE DESIGN. IF THE DESIGN OR THE ARCHITECTURE IS COMPROMISED (THE STANDARDS/RULES ARE BROKEN) THEN THE MODEL BECOMES INFLEXIBLE AND BRITTLE. BY BREAKING THE STANDARDS/RULES AND CHANGING THE ARCHITECTURE, RE-ENGINEERING BECOMES NECESSARY IN ORDER TO HANDLE BUSINESS CHANGES. ONCE THIS HAPPENS, TOTAL COST OF OWNERSHIP OVER THE LIFECYCLE OF THE DATA WAREHOUSE RISES, COMPLEXITY RISES, AND THE ENTIRE VALUE PROPOSITION OF APPLYING THE DATA VAULT CONCEPTS BREAKS DOWN.

For example, suppose a data warehouse is constructed to house parts – then after 3 months in operation the business would like to track suppliers. The Data Vault can quickly be adapted by adding a Supplier Hub, Supplier Satellites, followed by a Link table between parts and suppliers - the impact is minimal (if any) to existing loading routines and existing history held within.

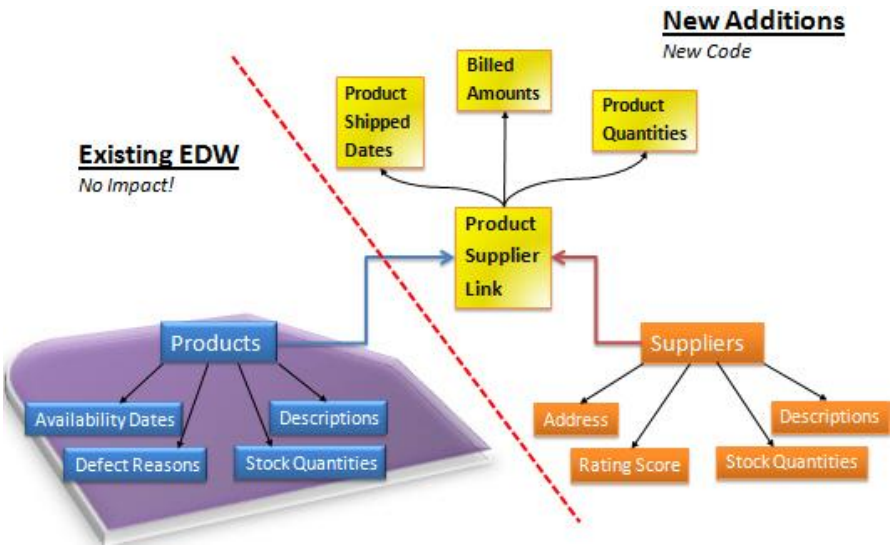


Figure 1-6: Flexibility of Adapting to Change

In Figure 1-6 we see the existing data warehouse in purple and the new sections in yellow and orange (to the right of the red dotted line). Placing the associations in a Link table enables the data warehouse design to be flexible in this manner – where new components do not affect existing components.

One difference between the Data Vault model and a 3rd normal form model is the use of Links to represent associations across concepts. 3rd Normal form represents most relationships by tying parent keys to child tables directly (without an extrapolated association table).

Relationships such as 1 to Many, Many to 1, and 1 to 1 are represented in 3rd normal form directly by embedding the parent fields in the child tables. ***This leads to inflexibility of the model.***

When the business rules change and the cardinality of the data must change to meet business needs, the model is altered as is the operational application using the model.

An example of a 3rd normal form model is shown below in Figure 1-7.

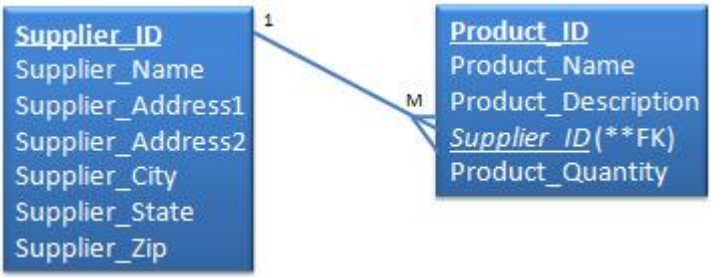


Figure 1-7: 3rd Normal Form Product and Supplier Example

In Figure 1-7, a Product can have 1 and only 1 supplier, but a Supplier can supply many products. With a model like this, the business rule may be: *"a Product can only be supplied by a single supplier"*, which means that the operational system that collects the information is coded accordingly. When or if the business changes its rule to say: *"a product can be supplied by more than one supplier"* then the application must change, as must the underlying data model structure.

While this appears to be a small change it may affect all kinds of underlying information in the operational system; especially if the product is a PARENT to other tables.

For data warehouses (except Data Vaults) this structure leads to even more complexity. In a data warehouse that contains foreign keys embedded in child tables, this leads to ***cascading change impacts.*** In other words, any changes made to parent keys will cascade ***all the way down*** in to every single child table. The end result?

- You have to rebuild ALL your ETL loading routines
- You have to rebuild ALL your Queries against the dependent structures
- You have to re-model ALL your parent and child tables

The end result is ***massive re-engineering efforts***, and that's not all! The problem gets exponentially harder to handle with larger and larger data warehouse models.

This is the #1 reason why Data Warehouse/BI Projects are "torn down, stopped, halted, burned, and ripped apart" or labeled failures! The growing and already high cost of re-engineering which is caused by poor architectural design and dependencies built in to your data warehouse model! Don't let this happen to you! Use a Data Vault and avoid this mess up-front.

This is evident when loading history to a data warehouse where the cardinality that exists in today's model did not exist in the past. Data warehousing teams run a high risk of re-engineering the architecture and loading process, if the enterprise data warehouse model enforces current relationships. The Data Vault model mitigates this risk by providing Link tables for all relationships regardless of cardinality.

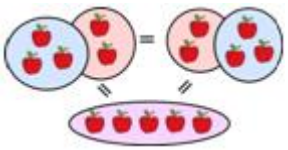
1.8 Data Vault Basis of Commutative Properties and Set Based Math

The Data Vault is based on raw data sets arriving at the warehouse (with little to no alteration of the data within). This is commonly referred to as "the raw data warehouse." There is a notion for constructing a business based Data Vault that will be discussed later in this book.

One of the founding principles of the Data Vault is: enable re-creation of a source system data for a specific point in time. The Data Vault achieves this by loading raw data, passively integrating it by business key and time-stamping the load cycles with the arrival dates of the data set.

There is a law in mathematics called the commutative property. The commutative property is defined below:

In mathematics, commutativity is the ability to change the order of something without changing the end result. It is a fundamental property in most branches of mathematics and many proofs depend on it. The commutativity of simple operations was for many years implicitly assumed and the property was not given a name or attributed until the 19th century when mathematicians began to formalize the theory of mathematics. http://en.wikipedia.org/wiki/Commutative



The basic notion is that A = B = C at a specific point in time, where A = a source system/source application, and B = staging area, and C = enterprise Data Vault; such that A can be reconstituted for any point in time contained within C. This preserves the auditability of the data set housed within the Data Vault – while offering base level integration across lines of business (see previous discussion on Hub based business keys).

In some cases B can represent the Data Vault while C represents "as-is raw level star schemas." Raw level star schemas are utilized to show the business what the source systems are collecting, and where the gaps may be between the business rules, business operations, and source system applications. Information quality (IQ) can be improved through resolution of the identified gaps.

To find out more about gap analysis, please read the book: The Next Business Supermodel. The Business of Data Vault Modeling.

Another founding principle behind the Data Vault architecture is the use of set logic or set based math. The Hubs and Links are loaded based on union sets of information, while the Satellites are loaded based on delta changes inclusive of the union functionality. Set logic is applied to the loading processes for restart ability, scalability, and partitioning of the components.

Standard set theory is defined as follows:

> Set theory, formalized using first-order logic, is the most common foundational system for mathematics. The language of set theory is used in the definitions of nearly all mathematical objects, such as functions, and concepts of set theory are integrated throughout the mathematics curriculum. Elementary facts about sets and set membership can be introduced in primary school, along with Venn diagrams, to study collections of commonplace physical objects. Elementary operations such as set union and intersection can be studied in this context. More advanced concepts such as cardinality are a standard part of the undergraduate mathematics curriculum. http://en.wikipedia.org/wiki/Set_theory

In the Data Vault approach, set theory is applied to incoming data sets. The set theory applied in loading routines is depicted in Figure 1-8:

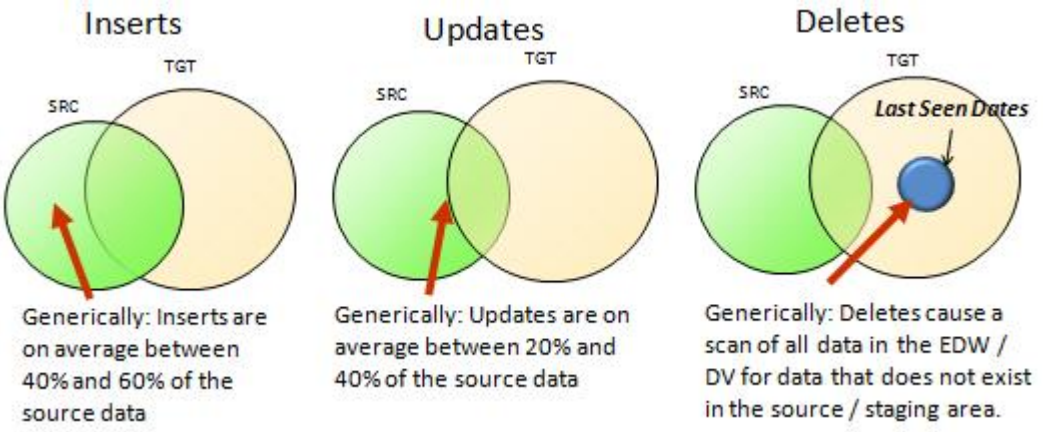


Figure 1-8: Applied Set Theory for the Data Vault

The set theory is applied again for Hub and Link loading where only new data (not previously inserted) is applied or loaded. Set-based logic is applied when single distinct lists of keys are loaded in order to the target table where they haven't yet been loaded.

1.9 Data Vault and Parallel Processing Mathematics

The main purpose to introducing set-theory concepts and the mathematics behind the Data Vault is to provide you with a glimpse of the actual engineering effort behind the Data Vault architecture itself. The architecture is not merely "just another design" of tables strung together – no, it is engineered with specific tolerance levels so that it can scale, and be flexible as necessary. These concepts are foundational to understanding why the architecture is designed, and what the specific purposes of the design elements are. I hope you find this section enlightening, as it explains some of the background reasons as to why you should stick with the original structures (unmodified) as you implement your Data Vault.

The Data Vault Modeling components are based on parallel processing mathematics (versus serial processing). Massively Parallel Systems typically use a shared-nothing design. The Data Vault Modeling components make use of this design technique to split data in a vertical format: aka vertical partitioning. The vertical partitioning of data is applied to Hub, Link and Satellite structures and is a base part of the architecture. The objective of vertical partitioning within the Data Vault Model is to split the work, so that the database engines can optimize the following:

- Index Coverage
- Data Redundancy (minimize this)
- Parallel Query
- Resource Utilization (split over hardware platforms)

If you are not familiar with the mathematical principles of Massively Parallel Processing (MPP) you can read about generic parallel processing rules and performance speed up (tasking) on Wikipedia. Figure 1-9 is a graph drawn from Wikipedia that introduces the concepts of Parallel Processing:

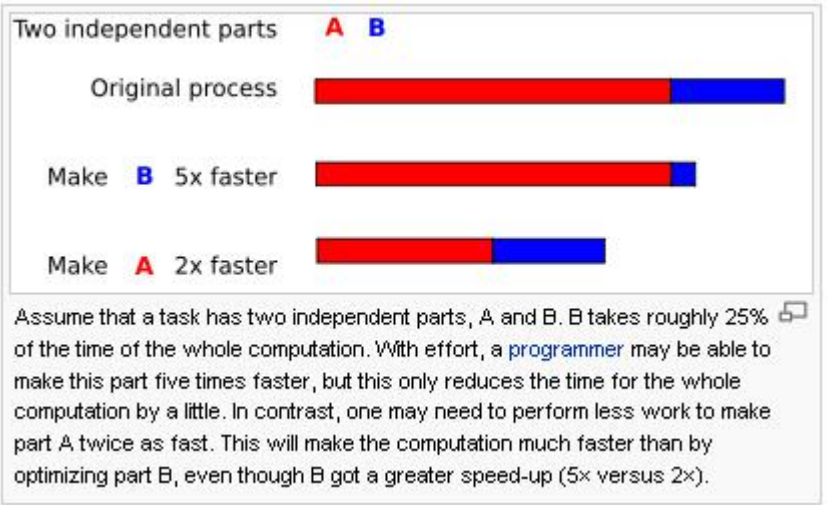


Figure 1-9: Parallel Computing Simplified

Image: http://en.wikipedia.org/wiki/Parallel_computing

The principles at work express themselves in the form of design and processing. The mathematics behind the Data Vault Model can be found by reading about parallel processing.

Specifically: Parallel Data Processing, Parallel Task Processing, and MPP systems design and architecture.

The topology of the computing cluster (database engine) can be any of the desired pieces including: star, ring, tree, hyper-cube, fat hyper-cube, or n-dimensional mesh. The Data Vault splits out the Business Keys, the relationships (associations), and the descriptive data (repetitive).

Business keys by nature are *generally* non-repetitive, therefore increasing index coverage dramatically. Business keys by nature are also specific to a *set* or *tuple* of data as an identifying marker. The Hubs therefore act as independent sources of information, making it easy (for instance) to split different Hubs across different computing platforms – in other words, applying vertical partitioning at the hardware level.

This is the nature of MPP and is known as "scale-out." Scale-out technology allows the model to grow as large as the data set or the business demands while keeping near linear performance gains (in relation to the scale of the hardware). The Links or associations also follow across multiple platforms, and are sometimes replicated in shared-nothing environments for ease of joins. One term for this is: "join-indexes".

This is just one way to view the Data Vault Model; it is essentially based on the principles of a scale-free tree, all the way down to the individual table structures built within the model. Multiple scale-free trees are nothing more than more Hubs, Links, and Satellites within the Data Vault, thus producing a "cube-like" structure if desired. An example of a logical design or conceptual view of the Data Vault in a Hyper Cube – it might look something like this:

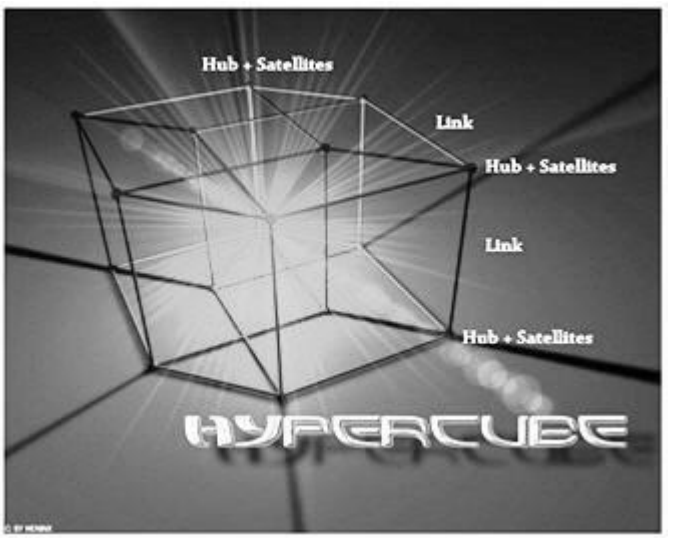


Figure 1-10: Logical Data Vault Hyper Cube

Image: http://clanbase.ggl.com/img.php?url=fc07.deviantart.com/fs14/i/2007/076/a/b/Hypercube_by_Meninx.jpg

In the physical model, Hubs are connected to Link structures; Links become a physical notion for an association. In the physical Data Vault Model nodes are connected through Links to each other, they are not directly related. This is a conceptual basis for establishing the premise of the vision. Hubs provide the keys, while Satellites around Hubs "describe" the key for any given

point in time. Hyper Cubes can be created as can trees. A simpler vision or view of the Data Vault Model split for parallelism is in Figure 1-11:

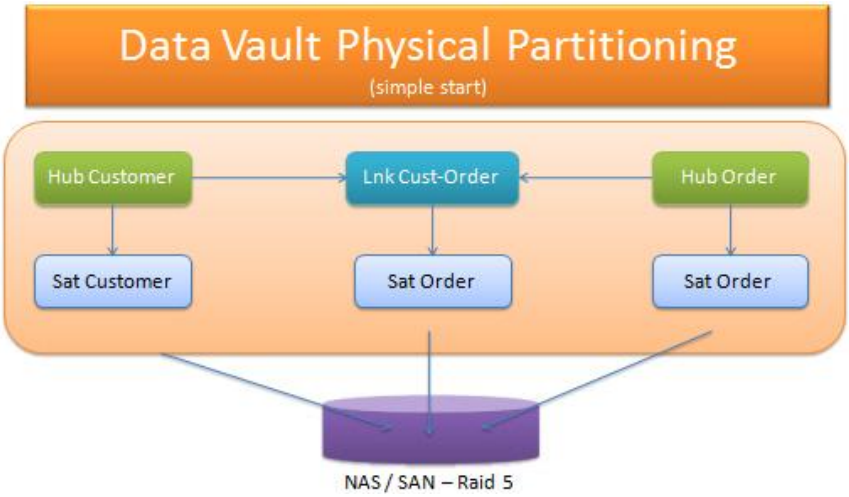


Figure 1-11: Physical Data Vault Layout (Starting point)

This is where it starts, quite simple enough – no real partitioning of the data because the size of the data set is not yet large enough. All of the tables go through one, two, or three I/O connections to a SAN or a NAS drive. When the data set grows, physical partitioning (or split-off of tables) can occur. The end-result (to an extreme) might be as shown in Figure 1-12:

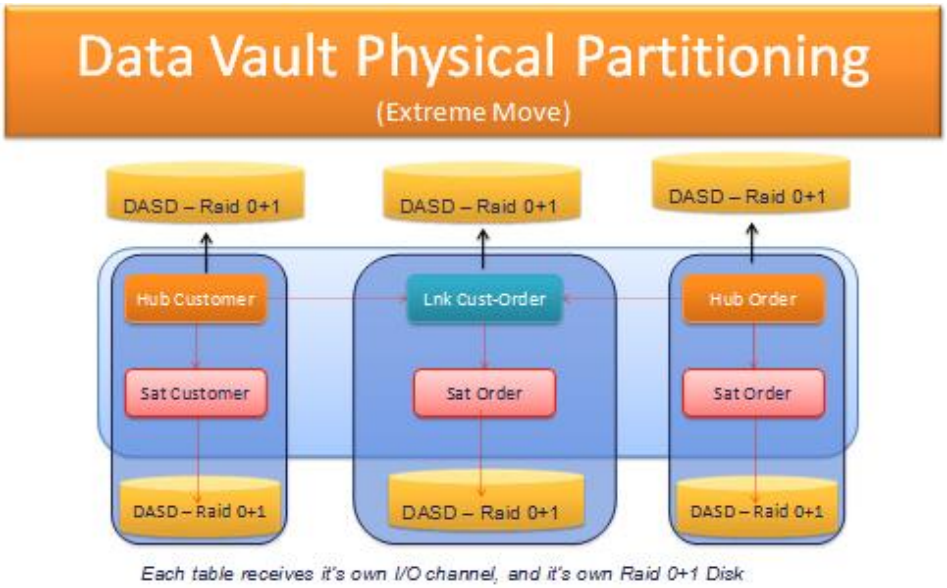


Figure 1-12: Physical Data Vault Layout (Partitioned)

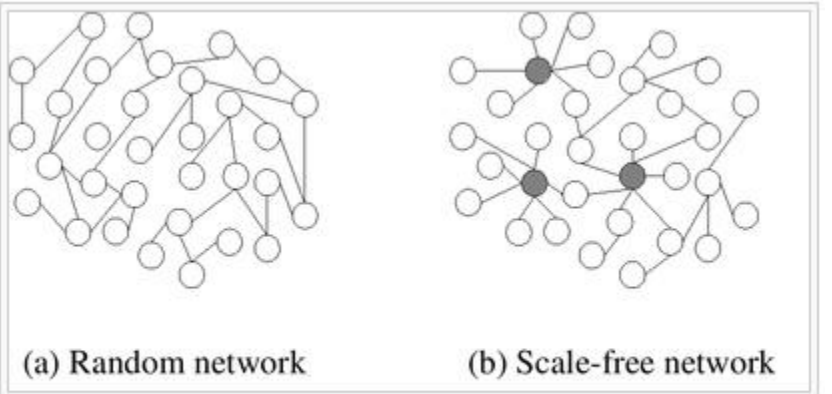
In this case, each table has I/O channels bound to it, along with dedicated disk (DASD) sitting on Raid 0+1 format. This is the ultimate in separation for relational database engines. This allows the relational database engines to operate their "parallel query engines" without disk wait

state dependencies across each table. Truly independent hardware levels can achieve very high performance. The next step might be separating the processing power out into different nodes, reaching the MPP level of architecture in hardware.

The Data Vault Model follows a scale-free topology. Scale-free topology is defined as follows:

A scale-free network is a network whose degree distribution follows a power law, at least asymptotically. That is, the fraction P(k) of nodes in the network having k connections to other nodes goes for large values of k as P(k) ~ k-? where ? is a constant whose value is typically in the range 2 < ? < 3, although occasionally it may lie outside these bounds.

Scale-free networks are noteworthy because many empirically observed networks appear to be scale-free, including the protein networks, citation networks, and some social networks.[1]



Random network (a) and scale-free network (b). In the scale-free network, the larger hubs are highlighted.

Source: ***http://en.wikipedia.org/wiki/Scale-free_network***

The mathematics behind scale-free networks applies to the Data Vault Model. Any physical model built from the Data Vault principles will carry the same mathematical properties. Using a spring-graph, or a weighted graph in either 2 dimensions or 3 dimensions, it becomes apparent which of the nodes are the "most important" in the Data Model. The most interconnected nodes are centralized within the graph, and have the most neighbors.

1.10 Introduction to Complexity and the Data Vault

This topic is really deserving of an entire chapter, perhaps even an entire book. However, in the interest of time, and due to the fact that this concept must be brought to light, I will make a small introduction to this concept. The Data Vault model and methodology make a tremendous contribution to lowering the overall complexity of the systems involved in data warehousing.

Current data warehousing systems try to do ***too much*** in their loading cycle. They try to address ALL of the following problems in a ***single load pattern***:

- Sourcing Problems:
  - Synchronization / Source Data Availability time windows
  - Cross-System Joins
  - Cross-System Filters
  - Cross-System aggregates
  - Indexing issues, leading to performance problems
  - Disjoint or missing source data sets
  - Missing source keys
  - Bad source data, out of range source data
  - Source system password issues
  - Source system Availability for loading windows
  - Source system CPU, RAM, and Disk Load
  - Source System structure complexity
  - Source system I/O performance
  - Source System transactional record locks
- Transformation problems – often IN STREAM
  - Cleansing
  - Quality and Alignment
  - Joins
  - Consolidation
  - Aggregation
  - Filtering
  - Sequence Assignment - often leading to lack of parallelism
  - Data type correction
  - Error handling (when the database kicks it back)
  - Error handling (data is: out of bounds, out of range)
  - Size of Memory
  - Lookup issues (more sourcing problems, caching problems, Memory problems)
  - Sorting issues (large caches, disk overflows, huge keys)
  - ***BUSINESS RULES***, especially across SPLIT data streams
  - Multiple targets
  - Multiple target errors
  - Multiple sources
  - Single transformation bottleneck (performance, realationships – joins, and so on)

- Target Problems
  - Lack of database tuning
  - Index updates (deadlocking)
  - Update, Insert, and Delete mixed statements – forcing data ORDER to be specific, cutting off possibilities for executing in parallel
  - Block size issues
  - Multi-target issues (too many connections, error handling in one stream holding up all other targets in the same data stream)
  - WIDE targets (due to business rules being IN-STREAM)
  - Indexes ON TARGETS (because targets ARE the data marts)
  - Lack of control over target partitioning

Along with many more issues. In these cases, this is the traditional view of issues that data integration specialists are left to solve. You are expected to construct load after load that answers ALL of these problems in a SINGLE data stream right? Well, this is no way to do business. This increases complexity to an unimaginable level, and this contributes to the ultimate downfall of the data warehousing project!

"Quality Software Management" Vol. 1 Gerald M. Weinberg pp. 135-139.
When you develop your ETL for a star schema EDW, you essentially get a sequential set of (big T) transformations. As that sequence grows in size and complexity, the difficulty of testing it, and tracing errors back to the source grows exponentially, hence as your (S-S) EDW grows, you get haunted by ever growing development cycles, and increasingly less control over the testing process, until your EDW has developed into yet another legacy system. And then you know what its fate will be…

The Data Vault Motto: DIVIDE AND CONQUER!

Believe me, you can ***win every time*** with this strategy. Let's analyze this for a minute:

Sourcing Problems: Nearly every problem can be addressed through a few simple rules:

- Separate each source load, and land the data in the target – make each load a very simple ***copy operation*** where the data is pulled from the source, and landed directly in the target – in this case, a STAGING AREA (as defined in the next chapter). Yes, you may source a CDC (Change Data Capture) operation if you so desire.
- Run the staging loads ***when the data is ready!*** Don't wait for other systems, don't perform any other systems joins, and don't force the data to "conform or align" with specific rules or data types.

These two simple rules ensure that you can get in, get the data from the source, and get out – when the source data is ready to go. No waiting! No Joining! No timing complexity! No performance problems! You can always take a copy operation and partition the target, and partition the load for MAXIMUM throughput!

Transformation problems: Divide and conquer. The following rules make it much easier to deal with this part of the loading cycle:

- Move the business rules downstream. This includes all the joins, filters, aggregations, quality, cleansing, and alignments that need to happen – between the Data Vault and the Data Marts. This also allows you to effectively target the PROPER data mart with the PROPER rule set (as deemed appropriate by the business).
- Load ***raw data*** in to the Data Vault area, this provides SIMPLE, maintainable, and easy to use loading code that meets the needs of the business. It also prevents you from having to "re-engineer" loading routines to add new systems, or add new data. Sure you end up with a lot more routines, BUT each one is a thousand times less complex, and easier to manage.

The end result for this? You can PARALLELIZE the loading routines to your data warehouse AND you can load data to your Data Vault in REAL-TIME at the SAME TIME as your batch loads are running. Just try that with your standard star-schema!

Targeting problems: They all but disappear. Why? Because once you divide and conquer, your loading routines will be built for inserts only, high speed inserts at that, and they generally will contain only one or two target tables for loading purposes! No more locking problems, no more worries about wide rows (except when you get to loading data marts, that's another story). High degrees of parallelism, high degrees of partitioning, and high performance, and ***really low complexity scores,*** what more could you ask for?

1.11 Loading Processes: Batch Versus Real Time

This book introduces the concepts with a small bit of background, it is meant to be only an introduction to the loading patterns and processes used within the Data Vault. The purpose of this entry is to define the basic terms of batch loading and real-time loading.

Batch Loading: usually occurring on a scheduled basis, loading any number of rows in a "batch". The execution timing will vary from every 5 minutes to every 24 hours, to weekly, to monthly, and so on. Any load cycle running every 20 seconds or less, tends to fall close to the real-time loading category. All other scheduled cycles tend to be labeled "mini-batches."

Real-Time Loading: there is a grey area of definition between what a batch load is and what a real-time load. For the purposes of this book, real-time loading is any loading cycle that runs continuously (never ends), loads data from a web-service or queuing service (usually) whenever the transactions appear.

Neither loading paradigm has any effect on the data modeling constructs within the Data Vault. The Hub, Link, and Satellite definitions remain the same and are capable of handling extremely large batches of data, and or extremely fast (millisecond feeds) loads of data.