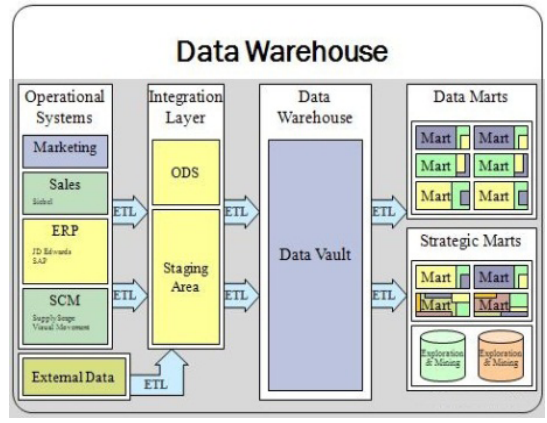
4

Understanding Data Warehousing

Data warehouse is the core of business intelligence. It is majorly used for reporting and analyzing data. Data mart, master data management, dimension, slowly changing dimension and star schema. This text elucidates the crucial theories and principles of data warehousing.

Data Warehouse

In computing, a data warehouse (DW or DWH), also known as an enterprise data warehouse (EDW), is a system used for reporting and data analysis, and is considered a core component of business intelligence. DWs are central repositories of integrated data from one or more disparate sources. They store current and historical data and are used for creating analytical reports for knowledge workers throughout the enterprise. Examples of reports could range from annual and quarterly comparisons and trends to detailed daily sales analysis.



Data Warehouse Overview

The data stored in the warehouse is uploaded from the operational systems (such as marketing or sales). The data may pass through an operational data store for additional operations before it is used in the DW for reporting.

Types of Systems

Data Mart

> A data mart is a simple form of a data warehouse that is focused on a single subject (or functional area), hence they draw data from a limited number of sources such as sales, finance or marketing. Data marts are often built and controlled by a single department within an organization. The sources could be internal operational systems, a central data

warehouse, or external data. Denormalization is the norm for data modeling techniques in this system. Given that data marts generally cover only a subset of the data contained in a data warehouse, they are often easier and faster to implement.

Difference between data warehouse and data mart	
Data warehouse	Data mart
enterprise-wide data	department-wide data
multiple subject areas	single subject area
difficult to build	easy to build
takes more time to build	less time to build
larger memory	limited memory

Types of Data Marts

- Dependent data mart
- Independent data mart
- Hybrid data mart

Online analytical processing (OLAP)

OLAP is characterized by a relatively low volume of transactions. Queries are often very complex and involve aggregations. For OLAP systems, response time is an effectiveness measure. OLAP applications are widely used by Data Mining techniques. OLAP databases store aggregated, historical data in multi-dimensional schemas (usually star schemas). OLAP systems typically have data latency of a few hours, as opposed to data marts, where latency is expected to be closer to one day. The OLAP approach is used to analyze multidimensional data from multiple sources and perspectives. The three basic operations in OLAP are : Roll-up (Consolidation), Drill-down and Slicing & Dicing.

Online transaction processing (OLTP)

OLTP is characterized by a large number of short on-line transactions (INSERT, UPDATE, DELETE). OLTP systems emphasize very fast query processing and maintaining data integrity in multi-access environments. For OLTP systems, effectiveness is measured by the number of transactions per second. OLTP databases contain detailed and current data. The schema used to store transactional databases is the entity model (usually 3NF). Normalization is the norm for data modeling techniques in this system.

Predictive analysis

Predictive analysis is about finding and quantifying hidden patterns in the data using complex mathematical models that can be used to predict future outcomes. Predictive analysis is different from OLAP in that OLAP focuses on historical data analysis and is reactive in nature, while predictive analysis focuses on the future. These systems are also used for CRM (customer relationship management).

Software Tools

The typical extract-transform-load (ETL)-based data warehouse uses staging, data integration, and access layers to house its key functions. The staging layer or staging database stores raw data extracted from each of the disparate source data systems. The integration layer integrates the disparate data sets by transforming the data from the staging layer often storing this transformed data in an operational data store (ODS) database. The integrated data are then moved to yet another database, often called the data warehouse database, where the data is arranged into hierarchical groups often called dimensions and into facts and aggregate facts. The combination of facts and dimensions is sometimes called a star schema. The access layer helps users retrieve data.

This definition of the data warehouse focuses on data storage. The main source of the data is cleaned, transformed, cataloged and made available for use by managers and other business professionals for data mining, online analytical processing, market research and decision support. However, the means to retrieve and analyze data, to extract, transform and load data, and to manage the data dictionary are also considered essential components of a data warehousing system. Many references to data warehousing use this broader context. Thus, an expanded definition for data warehousing includes business intelligence tools, tools to extract, transform and load data into the repository, and tools to manage and retrieve metadata.

Benefits

A data warehouse maintains a copy of information from the source transaction systems. This architectural complexity provides the opportunity to:

- Integrate data from multiple sources into a single database and data model. Mere congregation of data to single database so a single query engine can be used to present data is an ODS.
- Mitigate the problem of database isolation level lock contention in transaction processing systems caused by attempts to run large, long running, analysis queries in transaction processing databases.
- Maintain data history, even if the source transaction systems do not.
- Integrate data from multiple source systems, enabling a central view across the enterprise. This benefit is always valuable, but particularly so when the organization has grown by merger.
- Improve data quality, by providing consistent codes and descriptions, flagging or even fixing bad data.
- Present the organization's information consistently.
- Provide a single common data model for all data of interest regardless of the data's source.
- Restructure the data so that it makes sense to the business users.
- Restructure the data so that it delivers excellent query performance, even for complex an-

alytic queries, without impacting the operational systems.

- Add value to operational business applications, notably customer relationship management (CRM) systems.
- Make decision–support queries easier to write.
- Optimized data warehouse architectures allow data scientists to organize and disambiguate repetitive data.

Generic Environment

The environment for data warehouses and marts includes the following:

- Source systems that provide data to the warehouse or mart;
- Data integration technology and processes that are needed to prepare the data for use;
- Different architectures for storing data in an organization's data warehouse or data marts;
- Different tools and applications for the variety of users;
- Metadata, data quality, and governance processes must be in place to ensure that the warehouse or mart meets its purposes.

In regards to source systems listed above, Rainer states, "A common source for the data in data warehouses is the company's operational databases, which can be relational databases".

Regarding data integration, Rainer states, "It is necessary to extract data from source systems, transform them, and load them into a data mart or warehouse".

Rainer discusses storing data in an organization's data warehouse or data marts.

Metadata are data about data. "IT personnel need information about data sources; database, table, and column names; refresh schedules; and data usage measures".

Today, the most successful companies are those that can respond quickly and flexibly to market changes and opportunities. A key to this response is the effective and efficient use of data and information by analysts and managers. A "data warehouse" is a repository of historical data that are organized by subject to support decision makers in the organization. Once data are stored in a data mart or warehouse, they can be accessed.

History

The concept of data warehousing dates back to the late 1980s when IBM researchers Barry Devlin and Paul Murphy developed the "business data warehouse". In essence, the data warehousing concept was intended to provide an architectural model for the flow of data from operational systems to decision support environments. The concept attempted to address the various problems associated with this flow, mainly the high costs associated with it. In the absence of a data warehousing architecture, an enormous amount of redundancy was required to support multiple decision support environments. In larger corporations it was typical for multiple decision support envi-

ronments to operate independently. Though each environment served different users, they often required much of the same stored data. The process of gathering, cleaning and integrating data from various sources, usually from long-term existing operational systems (usually referred to as legacy systems), was typically in part replicated for each environment. Moreover, the operational systems were frequently reexamined as new decision support requirements emerged. Often new requirements necessitated gathering, cleaning and integrating new data from "data marts" that were tailored for ready access by users.

Key developments in early years of data warehousing were:

- 1960s – General Mills and Dartmouth College, in a joint research project, develop the terms *dimensions* and *facts*.
- 1970s – ACNielsen and IRI provide dimensional data marts for retail sales.
- 1970s – Bill Inmon begins to define and discuss the term: Data Warehouse.
- 1975 – Sperry Univac introduces MAPPER (MAintain, Prepare, and Produce Executive Reports) is a database management and reporting system that includes the world's first 4GL. First platform designed for building Information Centers (a forerunner of contemporary Enterprise Data Warehousing platforms)
- 1983 – Teradata introduces a database management system specifically designed for decision support.
- 1984 – Metaphor Computer Systems, founded by David Liddle and Don Massaro, releases Data Interpretation System (DIS). DIS was a hardware/software package and GUI for business users to create a database management and analytic system.
- 1988 – Barry Devlin and Paul Murphy publish the article *An architecture for a business and information system* where they introduce the term "business data warehouse".
- 1990 – Red Brick Systems, founded by Ralph Kimball, introduces Red Brick Warehouse, a database management system specifically for data warehousing.
- 1991 – Prism Solutions, founded by Bill Inmon, introduces Prism Warehouse Manager, software for developing a data warehouse.
- 1992 – Bill Inmon publishes the book *Building the Data Warehouse*.
- 1995 – The Data Warehousing Institute, a for-profit organization that promotes data warehousing, is founded.
- 1996 – Ralph Kimball publishes the book *The Data Warehouse Toolkit*.
- 2012 – Bill Inmon developed and made public technology known as "textual disambiguation". Textual disambiguation applies context to raw text and reformats the raw text and context into a standard data base format. Once raw text is passed through textual disambiguation, it can easily and efficiently be accessed and analyzed by standard business intelligence technology. Textual disambiguation is accomplished through the execution of textual ETL. Textual disambiguation is useful wherever raw text is found, such as in documents, Hadoop, email, and so forth.

Information Storage

Facts

A fact is a value or measurement, which represents a fact about the managed entity or system.

Facts as reported by the reporting entity are said to be at raw level. E.g. in a mobile telephone system, if a BTS (base transceiver station) received 1,000 requests for traffic channel allocation, it allocates for 820 and rejects the remaining then it would report 3 facts or measurements to a management system:

- tch_req_total = 1000
- tch_req_success = 820
- tch_req_fail = 180

Facts at the raw level are further aggregated to higher levels in various dimensions to extract more service or business-relevant information from it. These are called aggregates or summaries or aggregated facts.

For instance, if there are 3 BTSs in a city, then the facts above can be aggregated from the BTS to the city level in the network dimension. For example:

- $tch\_req\_success\_city = tch\_req\_success\_bts1 + tch\_req\_success\_bts2 + tch\_req\_success\_bts3$
- $avg\_tch\_req\_success\_city = (tch\_req\_success\_bts1 + tch\_req\_success\_bts2 + tch\_req\_success\_bts3) / 3$

Dimensional Versus Normalized Approach for Storage of Data

There are three or more leading approaches to storing data in a data warehouse — the most important approaches are the dimensional approach and the normalized approach.

The dimensional approach refers to Ralph Kimball's approach in which it is stated that the data warehouse should be modeled using a Dimensional Model/star schema. The normalized approach, also called the 3NF model (Third Normal Form) refers to Bill Inmon's approach in which it is stated that the data warehouse should be modeled using an E-R model/normalized model.

In a dimensional approach, transaction data are partitioned into "facts", which are generally numeric transaction data, and "dimensions", which are the reference information that gives context to the facts. For example, a sales transaction can be broken up into facts such as the number of products ordered and the total price paid for the products, and into dimensions such as order date, customer name, product number, order ship-to and bill-to locations, and salesperson responsible for receiving the order.

A key advantage of a dimensional approach is that the data warehouse is easier for the user to understand and to use. Also, the retrieval of data from the data warehouse tends to operate very quickly. Dimensional structures are easy to understand for business users, because the structure is divided into measurements/facts and context/dimensions. Facts are related to the organization's business processes and operational system whereas the dimensions surrounding them contain context about the measurement (Kimball, Ralph 2008). Another advantage offered by dimension-

al model is that it does not involve a relational database every time. Thus, this type of modeling technique is very useful for end-user queries in data warehouse.

The main disadvantages of the dimensional approach are the following:

1. In order to maintain the integrity of facts and dimensions, loading the data warehouse with data from different operational systems is complicated.
2. It is difficult to modify the data warehouse structure if the organization adopting the dimensional approach changes the way in which it does business.

In the normalized approach, the data in the data warehouse are stored following, to a degree, database normalization rules. Tables are grouped together by *subject areas* that reflect general data categories (e.g., data on customers, products, finance, etc.). The normalized structure divides data into entities, which creates several tables in a relational database. When applied in large enterprises the result is dozens of tables that are linked together by a web of joins. Furthermore, each of the created entities is converted into separate physical tables when the database is implemented (Kimball, Ralph 2008). The main advantage of this approach is that it is straightforward to add information into the database. Some disadvantages of this approach are that, because of the number of tables involved, it can be difficult for users to join data from different sources into meaningful information and to access the information without a precise understanding of the sources of data and of the data structure of the data warehouse.

Both normalized and dimensional models can be represented in entity-relationship diagrams as both contain joined relational tables. The difference between the two models is the degree of normalization (also known as Normal Forms). These approaches are not mutually exclusive, and there are other approaches. Dimensional approaches can involve normalizing data to a degree (Kimball, Ralph 2008).

In *Information-Driven Business*, Robert Hillard proposes an approach to comparing the two approaches based on the information needs of the business problem. The technique shows that normalized models hold far more information than their dimensional equivalents (even when the same fields are used in both models) but this extra information comes at the cost of usability. The technique measures information quantity in terms of information entropy and usability in terms of the Small Worlds data transformation measure.

Design Methods

Bottom-up Design

In the *bottom-up* approach, data marts are first created to provide reporting and analytical capabilities for specific business processes. These data marts can then be integrated to create a comprehensive data warehouse. The data warehouse bus architecture is primarily an implementation of "the bus", a collection of conformed dimensions and conformed facts, which are dimensions that are shared (in a specific way) between facts in two or more data marts.

Top-down Design

The *top-down* approach is designed using a normalized enterprise data model. "Atomic" data, that

is, data at the greatest level of detail, are stored in the data warehouse. Dimensional data marts containing data needed for specific business processes or specific departments are created from the data warehouse.

Hybrid Design

Data warehouses (DW) often resemble the hub and spokes architecture. Legacy systems feeding the warehouse often include customer relationship management and enterprise resource planning, generating large amounts of data. To consolidate these various data models, and facilitate the extract transform load process, data warehouses often make use of an operational data store, the information from which is parsed into the actual DW. To reduce data redundancy, larger systems often store the data in a normalized way. Data marts for specific reports can then be built on top of the DW.

The DW database in a hybrid solution is kept on third normal form to eliminate data redundancy. A normal relational database, however, is not efficient for business intelligence reports where dimensional modelling is prevalent. Small data marts can shop for data from the consolidated warehouse and use the filtered, specific data for the fact tables and dimensions required. The DW provides a single source of information from which the data marts can read, providing a wide range of business information. The hybrid architecture allows a DW to be replaced with a master data management solution where operational, not static information could reside.

The Data Vault Modeling components follow hub and spokes architecture. This modeling style is a hybrid design, consisting of the best practices from both third normal form and star schema. The Data Vault model is not a true third normal form, and breaks some of its rules, but it is a top-down architecture with a bottom up design. The Data Vault model is geared to be strictly a data warehouse. It is not geared to be end-user accessible, which when built, still requires the use of a data mart or star schema based release area for business purposes.

Versus Operational System

Operational systems are optimized for preservation of data integrity and speed of recording of business transactions through use of database normalization and an entity-relationship model. Operational system designers generally follow the Codd rules of database normalization in order to ensure data integrity. Codd defined five increasingly stringent rules of normalization. Fully normalized database designs (that is, those satisfying all five Codd rules) often result in information from a business transaction being stored in dozens to hundreds of tables. Relational databases are efficient at managing the relationships between these tables. The databases have very fast insert/update performance because only a small amount of data in those tables is affected each time a transaction is processed. Finally, in order to improve performance, older data are usually periodically purged from operational systems.

Data warehouses are optimized for analytic access patterns. Analytic access patterns generally involve selecting specific fields and rarely if ever 'select *' as is more common in operational databases. Because of these differences in access patterns, operational databases (loosely, OLTP) benefit from the use of a row-oriented DBMS whereas analytics databases (loosely, OLAP) benefit from the use of a column-oriented DBMS. Unlike operational systems which maintain a snapshot

of the business, data warehouses generally maintain an infinite history which is implemented through ETL processes that periodically migrate data from the operational systems over to the data warehouse.

Evolution in Organization Use

These terms refer to the level of sophistication of a data warehouse:

Offline operational data warehouse

> Data warehouses in this stage of evolution are updated on a regular time cycle (usually daily, weekly or monthly) from the operational systems and the data is stored in an integrated reporting-oriented data

Offline data warehouse

> Data warehouses at this stage are updated from data in the operational systems on a regular basis and the data warehouse data are stored in a data structure designed to facilitate reporting.

On time data warehouse

> Online Integrated Data Warehousing represent the real time Data warehouses stage data in the warehouse is updated for every transaction performed on the source data

Integrated data warehouse

> These data warehouses assemble data from different areas of business, so users can look up the information they need across other systems.

Data Mart

A data mart is the access layer of the data warehouse environment that is used to get data out to the users. The data mart is a subset of the data warehouse and is usually oriented to a specific business line or team. Whereas data warehouses have an enterprise-wide depth, the information in data marts pertains to a single department. In some deployments, each department or business unit is considered the *owner* of its data mart including all the *hardware*, *software* and *data*. This enables each department to isolate the use, manipulation and development of their data. In other deployments where conformed dimensions are used, this business unit ownership will not hold true for shared dimensions like customer, product, etc.

Organizations build data warehouses and data marts because the information in the database is not organized in a way that makes it readily accessible, requiring queries that are too complicated or resource-consuming.

While transactional databases are designed to be updated, data warehouses or marts are read only. Data warehouses are designed to access large groups of related records. Data marts improve end-user response time by allowing users to have access to the specific type of data they need to view most often by providing the data in a way that supports the collective view of a group of users.

A data mart is basically a condensed and more focused version of a data warehouse that reflects the regulations and process specifications of each business unit within an organization. Each data mart is dedicated to a specific business function or region. This subset of data may span across many or all of an enterprise's functional subject areas. It is common for multiple data marts to be used in order to serve the needs of each individual business unit (different data marts can be used to obtain specific information for various enterprise departments, such as accounting, marketing, sales, etc.).

The related term spreadmart is a derogatory label describing the situation that occurs when one or more business analysts develop a system of linked spreadsheets to perform a business analysis, then grow it to a size and degree of complexity that makes it nearly impossible to maintain.

Data mart vs data warehouse

Data warehouse:

- Holds multiple subject areas
- Holds very detailed information
- Works to integrate all data sources
- Does not necessarily use a dimensional model but feeds dimensional models.

Data mart:

- Often holds only one subject area- for example, Finance, or Sales
- May hold more summarized data (although many hold full detail)
- Concentrates on integrating information from a given subject area or set of source systems
- Is built focused on a dimensional model using a star schema.

Design Schemas

- Star schema - fairly popular design choice; enables a relational database to emulate the analytical functionality of a multidimensional database
- Snowflake schema

Reasons for Creating a Data Mart

- Easy access to frequently needed data
- Creates collective view by a group of users
- Improves end-user response time
- Ease of creation
- Lower cost than implementing a full data warehouse
- Potential users are more clearly defined than in a full data warehouse
- Contains only business essential data and is less cluttered.

Dependent Data Mart

According to the Inmon school of data warehousing, a dependent data mart is a logical subset (view) or a physical subset (extract) of a larger data warehouse, isolated for one of the following reasons:

- A need refreshment for a special data model or schema: e.g., to restructure for OLAP
- Performance: to offload the data mart to a separate computer for greater efficiency or to eliminate the need to manage that workload on the centralized data warehouse.
- Security: to separate an authorized data subset selectively
- Expediency: to bypass the data governance and authorizations required to incorporate a new application on the Enterprise Data Warehouse
- Proving Ground: to demonstrate the viability and ROI (return on investment) potential of an application prior to migrating it to the Enterprise Data Warehouse
- Politics: a coping strategy for IT (Information Technology) in situations where a user group has more influence than funding or is not a good citizen on the centralized data warehouse.
- Politics: a coping strategy for consumers of data in situations where a data warehouse team is unable to create a usable data warehouse.

According to the Inmon school of data warehousing, tradeoffs inherent with data marts include limited scalability, duplication of data, data inconsistency with other silos of information, and inability to leverage enterprise sources of data.

The alternative school of data warehousing is that of Ralph Kimball. In his view, a data warehouse is nothing more than the union of all the data marts. This view helps to reduce costs and provides fast development, but can create an inconsistent data warehouse, especially in large organizations. Therefore, Kimball's approach is more suitable for small-to-medium corporations.

Master Data Management

In business, master data management (MDM) comprises the processes, governance, policies, standards and tools that consistently define and manage the critical data of an organization to provide a single point of reference.

The data that is mastered may include:

- reference data – the business objects for transactions, and the dimensions for analysis
- analytical data – supports decision making

In computing, a master data management tool can be used to support master data management by removing duplicates, standardizing data (mass maintaining), and incorporating rules to eliminate incorrect data from entering the system in order to create an authoritative source of master data. Master data are the products, accounts and parties for which the business transactions are

completed. The root cause problem stems from business unit and product line segmentation, in which the same customer will be serviced by different product lines, with redundant data being entered about the customer (aka party in the role of customer) and account in order to process the transaction. The redundancy of party and account data is compounded in the front to back office life cycle, where the authoritative single source for the party, account and product data is needed but is often once again redundantly entered or augmented.

Master data management has the objective of providing processes for collecting, aggregating, matching, consolidating, quality-assuring, persisting and distributing such data throughout an organization to ensure consistency and control in the ongoing maintenance and application use of this information.

The term recalls the concept of a *master file* from an earlier computing era.

Definition

Master data management (MDM) is a comprehensive method of enabling an enterprise to link all of its critical data to one file, called a master file, that provides a common point of reference. When properly done, master data management streamlines data sharing among personnel and departments. In addition, master data management can facilitate computing in multiple system architectures, platforms and applications.

At its core Master Data Management (MDM) can be viewed as a "discipline for specialized quality improvement" defined by the policies and procedures put in place by a data governance organization. The ultimate goal being to provide the end user community with a "trusted single version of the truth" from which to base decisions.

Issues

At a basic level, master data management seeks to ensure that an organization does not use multiple (potentially inconsistent) versions of the same master data in different parts of its operations, which can occur in large organizations. A typical example of poor master data management is the scenario of a bank at which a customer has taken out a mortgage and the bank begins to send mortgage solicitations to that customer, ignoring the fact that the person already has a mortgage account relationship with the bank. This happens because the customer information used by the marketing section within the bank lacks integration with the customer information used by the customer services section of the bank. Thus the two groups remain unaware that an existing customer is also considered a sales lead. The process of record linkage is used to associate different records that correspond to the same entity, in this case the same person.

Other problems include (for example) issues with the quality of data, consistent classification and identification of data, and data-reconciliation issues. Master data management of disparate data systems requires data transformations as the data extracted from the disparate source data system is transformed and loaded into the master data management hub. To synchronize the disparate source master data, the managed master data extracted from the master data management hub is again transformed and loaded into the disparate source data system as the master data is updated. As with other Extract, Transform, Load-based data movement, these processes are expensive and

inefficient to develop and to maintain which greatly reduces the return on investment for the master data management product.

One of the most common reasons some large corporations experience massive issues with master data management is growth through mergers or acquisitions. Any organizations which merge will typically create an entity with duplicate master data (since each likely had at least one master database of its own prior to the merger). Ideally, database administrators resolve this problem through deduplication of the master data as part of the merger. In practice, however, reconciling several master data systems can present difficulties because of the dependencies that existing applications have on the master databases. As a result, more often than not the two systems do not fully merge, but remain separate, with a special reconciliation process defined that ensures consistency between the data stored in the two systems. Over time, however, as further mergers and acquisitions occur, the problem multiplies, more and more master databases appear, and data-reconciliation processes become extremely complex, and consequently unmanageable and unreliable. Because of this trend, one can find organizations with 10, 15, or even as many as 100 separate, poorly integrated master databases, which can cause serious operational problems in the areas of customer satisfaction, operational efficiency, decision support, and regulatory compliance.

Solutions

Processes commonly seen in master data management include source identification, data collection, data transformation, normalization, rule administration, error detection and correction, data consolidation, data storage, data distribution, data classification, taxonomy services, item master creation, schema mapping, product codification, data enrichment and data governance.

The selection of entities considered for master data management depends somewhat on the nature of an organization. In the common case of commercial enterprises, master data management may apply to such entities as customer (customer data integration), product (product information management), employee, and vendor. Master data management processes identify the sources from which to collect descriptions of these entities. In the course of transformation and normalization, administrators adapt descriptions to conform to standard formats and data domains, making it possible to remove duplicate instances of any entity. Such processes generally result in an organizational master data management repository, from which all requests for a certain entity instance produce the same description, irrespective of the originating sources and the requesting destination.

The tools include data networks, file systems, a data warehouse, data marts, an operational data store, data mining, data analysis, data visualization, data federation and data virtualization. One of the newest tools, virtual master data management utilizes data virtualization and a persistent metadata server to implement a multi-level automated master data management hierarchy.

Transmission of Master Data

There are several ways in which master data may be collated and distributed to other systems. This includes:

- Data consolidation – The process of capturing master data from multiple sources and in-

tegrating into a single hub (operational data store) for replication to other destination systems.

- Data federation – The process of providing a single virtual view of master data from one or more sources to one or more destination systems.
- Data propagation – The process of copying master data from one system to another, typically through point-to-point interfaces in legacy systems.

Dimension (Data Warehouse)

A dimension is a structure that categorizes facts and measures in order to enable users to answer business questions. Commonly used dimensions are people, products, place and time.

In a data warehouse, dimensions provide structured labeling information to otherwise unordered numeric measures. The dimension is a data set composed of individual, non-overlapping data elements. The primary functions of dimensions are threefold: to provide filtering, grouping and labelling.

These functions are often described as "slice and dice". Slicing refers to filtering data. Dicing refers to grouping data. A common data warehouse example involves sales as the measure, with customer and product as dimensions. In each sale a customer buys a product. The data can be sliced by removing all customers except for a group under study, and then diced by grouping by product.

A dimensional data element is similar to a categorical variable in statistics.

Typically dimensions in a data warehouse are organized internally into one or more hierarchies. "Date" is a common dimension, with several possible hierarchies:

- "Days (are grouped into) Months (which are grouped into) Years",
- "Days (are grouped into) Weeks (which are grouped into) Years"
- "Days (are grouped into) Months (which are grouped into) Quarters (which are grouped into) Years"
- etc.

Types

Conformed Dimension

A conformed dimension is a set of data attributes that have been physically referenced in multiple database tables using the same key value to refer to the same structure, attributes, domain values, definitions and concepts. A conformed dimension cuts across many facts.

Dimensions are conformed when they are either exactly the same (including keys) or one is a perfect subset of the other. Most important, the row headers produced in two different answer sets from the same conformed dimension(s) must be able to match perfectly.

Conformed dimensions are either identical or strict mathematical subsets of the most granular, detailed dimension. Dimension tables are not conformed if the attributes are labeled differently or contain different values. Conformed dimensions come in several different flavors. At the most basic level, conformed dimensions mean exactly the same thing with every possible fact table to which they are joined. The date dimension table connected to the sales facts is identical to the date dimension connected to the inventory facts.

Junk Dimension

A junk dimension is a convenient grouping of typically low-cardinality flags and indicators. By creating an abstract dimension, these flags and indicators are removed from the fact table while placing them into a useful dimensional framework. A Junk Dimension is a dimension table consisting of attributes that do not belong in the fact table or in any of the existing dimension tables. The nature of these attributes is usually text or various flags, e.g. non-generic comments or just simple yes/no or true/false indicators. These kinds of attributes are typically remaining when all the obvious dimensions in the business process have been identified and thus the designer is faced with the challenge of where to put these attributes that do not belong in the other dimensions.

One solution is to create a new dimension for each of the remaining attributes, but due to their nature, it could be necessary to create a vast number of new dimensions resulting in a fact table with a very large number of foreign keys. The designer could also decide to leave the remaining attributes in the fact table but this could make the row length of the table unnecessarily large if, for example, the attributes is a long text string.

The solution to this challenge is to identify all the attributes and then put them into one or several Junk Dimensions. One Junk Dimension can hold several true/false or yes/no indicators that have no correlation with each other, so it would be convenient to convert the indicators into a more describing attribute. An example would be an indicator about whether a package had arrived, instead of indicating this as “yes” or “no”, it would be converted into “arrived” or “pending” in the junk dimension. The designer can choose to build the dimension table so it ends up holding all the indicators occurring with every other indicator so that all combinations are covered. This sets up a fixed size for the table itself which would be 2^x rows, where x is the number of indicators. This solution is appropriate in situations where the designer would expect to encounter a lot of different combinations and where the possible combinations are limited to an acceptable level. In a situation where the number of indicators are large, thus creating a very big table or where the designer only expect to encounter a few of the possible combinations, it would be more appropriate to build each row in the junk dimension as new combinations are encountered. To limit the size of the tables, multiple junk dimensions might be appropriate in other situations depending on the correlation between various indicators.

Junk dimensions are also appropriate for placing attributes like non-generic comments from the fact table. Such attributes might consist of data from an optional comment field when a customer places an order and as a result will probably be blank in many cases. Therefore, the junk dimension should contain a single row representing the blanks as a surrogate key that will be used in the fact table for every row returned with a blank comment field

Degenerate Dimension

A degenerate dimension is a key, such as a transaction number, invoice number, ticket number, or bill-of-lading number, that has no attributes and hence does not join to an actual dimension table. Degenerate dimensions are very common when the grain of a fact table represents a single transaction item or line item because the degenerate dimension represents the unique identifier of the parent. Degenerate dimensions often play an integral role in the fact table's primary key.

Role-playing Dimension

Dimensions are often recycled for multiple applications within the same database. For instance, a "Date" dimension can be used for "Date of Sale", as well as "Date of Delivery", or "Date of Hire". This is often referred to as a "role-playing dimension".

Use of ISO Representation Terms

When referencing data from a metadata registry such as ISO/IEC 11179, representation terms such as Indicator (a boolean true/false value), Code (a set of non-overlapping enumerated values) are typically used as dimensions. For example, using the National Information Exchange Model (NIEM) the data element name would be PersonGenderCode and the enumerated values would be male, female and unknown.

Dimension Table

In data warehousing, a dimension table is one of the set of companion tables to a fact table.

The fact table contains business facts (or *measures*), and foreign keys which refer to candidate keys (normally primary keys) in the dimension tables.

Contrary to *fact* tables, *dimension* tables contain descriptive attributes (or fields) that are typically textual fields (or discrete numbers that behave like text). These attributes are designed to serve two critical purposes: query constraining and/or filtering, and query result set labeling.

Dimension attributes should be:

- Verbose (labels consisting of full words)
- Descriptive
- Complete (having no missing values)
- Discretely valued (having only one value per dimension table row)
- Quality assured (having no misspellings or impossible values)

Dimension table rows are uniquely identified by a single key field. It is recommended that the key field be a simple integer because a key value is meaningless, used only for joining fields between the fact and dimension tables. Dimension tables often use primary keys that are also surrogate keys. Surrogate keys are often auto-generated (e.g. a Sybase or SQL Server "identity column", a PostgreSQL or Informix serial, an Oracle SEQUENCE or a column defined with AUTO_INCREMENT in MySQL).

The use of surrogate dimension keys brings several advantages, including:

- Performance. Join processing is made much more efficient by using a single field (the surrogate key)
- Buffering from operational key management practices. This prevents situations where removed data rows might reappear when their natural keys get reused or reassigned after a long period of dormancy
- Mapping to integrate disparate sources
- Handling unknown or not-applicable connections
- Tracking changes in dimension attribute values

Although surrogate key use places a burden put on the ETL system, pipeline processing can be improved, and ETL tools have built-in improved surrogate key processing.

The goal of a dimension table is to create standardized, conformed dimensions that can be shared across the enterprise's data warehouse environment, and enable joining to multiple fact tables representing various business processes.

Conformed dimensions are important to the enterprise nature of DW/BI systems because they promote:

- Consistency. Every fact table is filtered consistently, so that query answers are labeled consistently.
- Integration. Queries can drill into different process fact tables separately for each individual fact table, then join the results on common dimension attributes.
- Reduced development time to market. The common dimensions are available without recreating them.

Over time, the attributes of a given row in a dimension table may change. For example, the shipping address for a company may change. Kimball refers to this phenomenon as Slowly Changing Dimensions. Strategies for dealing with this kind of change are divided into three categories:

- Type One. Simply overwrite the old value(s).
- Type Two. Add a new row containing the new value(s), and distinguish between the rows using Tuple-versioning techniques.
- Type Three. Add a new attribute to the existing row.

Common Patterns

Date and time

Since many fact tables in a data warehouse are time series of observations, one or more date dimensions are often needed. One of the reasons to have date dimensions is to place calendar knowledge in the data warehouse instead of hard coded in an application. While a simple SQL date/timestamp is useful for providing accurate information about the time a fact was recorded, it can

not give information about holidays, fiscal periods, etc. An SQL date/timestamp can still be useful to store in the fact table, as it allows for precise calculations.

Having both the date and time of day in the same dimension, may easily result in a huge dimension with millions of rows. If a high amount of detail is needed it is usually a good idea to split date and time into two or more separate dimensions. A time dimension with a grain of seconds in a day will only have 86400 rows. A more or less detailed grain for date/time dimensions can be chosen depending on needs. As examples, date dimensions can be accurate to year, quarter, month or day and time dimensions can be accurate to hours, minutes or seconds.

As a rule of thumb, time of day dimension should only be created if hierarchical groupings are needed or if there are meaningful textual descriptions for periods of time within the day (ex. "evening rush" or "first shift").

If the rows in a fact table are coming from several timezones, it might be useful to store date and time in both local time and a standard time. This can be done by having two dimensions for each date/time dimension needed – one for local time, and one for standard time. Storing date/time in both local and standard time, will allow for analysis on when facts are created in a local setting and in a global setting as well. The standard time chosen can be a global standard time (ex. UTC), it can be the local time of the business' headquarter, or any other time zone that would make sense to use.

Slowly Changing Dimension

Dimensions in data management and data warehousing contain relatively static data about such entities as geographical locations, customers, or products. Data captured by Slowly Changing Dimensions (SCDs) change slowly but unpredictably, rather than according to a regular schedule.

Some scenarios can cause Referential integrity problems.

For example, a database may contain a fact table that stores sales records. This fact table would be linked to dimensions by means of foreign keys. One of these dimensions may contain data about the company's salespeople: e.g., the regional offices in which they work. However, the salespeople are sometimes transferred from one regional office to another. For historical sales reporting purposes it may be necessary to keep a record of the fact that a particular sales person had been assigned to a particular regional office at an earlier date, whereas that sales person is now assigned to a different regional office.

Dealing with these issues involves SCD management methodologies referred to as Type 0 through 6. Type 6 SCDs are also sometimes called Hybrid SCDs.

Type 0: Retain Original

The Type 0 method is passive. It manages dimensional changes and no action is performed. Values remain as they were at the time the dimension record was first inserted. In certain circumstances history is preserved with a Type 0. Higher order types are employed to guarantee the preservation of history whereas Type 0 provides the least or no control. This is rarely used.

Type 1: Overwrite

This methodology overwrites old with new data, and therefore does not track historical data.

Example of a supplier table:

Supplier_Key	Supplier_Code	Supplier_Name	Supplier_State
123	ABC	Acme Supply Co	CA

In the above example, Supplier_Code is the natural key and Supplier_Key is a surrogate key. Technically, the surrogate key is not necessary, since the row will be unique by the natural key (Supplier_Code). However, to optimize performance on joins use integer rather than character keys (unless the number of bytes in the character key is less than the number of bytes in the integer key).

If the supplier relocates the headquarters to Illinois the record would be overwritten:

Supplier_Key	Supplier_Code	Supplier_Name	Supplier_State
123	ABC	Acme Supply Co	IL

The disadvantage of the Type 1 method is that there is no history in the data warehouse. It has the advantage however that it's easy to maintain.

If one has calculated an aggregate table summarizing facts by state, it will need to be recalculated when the Supplier_State is changed.

Type 2: Add New Row

This method tracks historical data by creating multiple records for a given natural key in the dimensional tables with separate surrogate keys and/or different version numbers. Unlimited history is preserved for each insert.

For example, if the supplier relocates to Illinois the version numbers will be incremented sequentially:

Supplier_Key	Supplier_Code	Supplier_Name	Supplier_State	Version.
123	ABC	Acme Supply Co	CA	0
124	ABC	Acme Supply Co	IL	1

Another method is to add 'effective date' columns.

Supplier_Key	Supplier_Code	Supplier_Name	Supplier_State	Start_Date	End_Date
123	ABC	Acme Supply Co	CA	01-Jan-2000	21-Dec-2004
124	ABC	Acme Supply Co	IL	22-Dec-2004	NULL

The null End_Date in row two indicates the current tuple version. In some cases, a standardized surrogate high date (e.g. 9999-12-31) may be used as an end date, so that the field can be included in an index, and so that null-value substitution is not required when querying.

Transactions that reference a particular surrogate key (Supplier_Key) are then permanently bound to the time slices defined by that row of the slowly changing dimension table. An aggregate table

summarizing facts by state continues to reflect the historical state, i.e. the state the supplier was in at the time of the transaction; no update is needed. To reference the entity via the natural key, it is necessary to remove the unique constraint making Referential integrity by DBMS impossible.

If there are retroactive changes made to the contents of the dimension, or if new attributes are added to the dimension (for example a Sales_Rep column) which have different effective dates from those already defined, then this can result in the existing transactions needing to be updated to reflect the new situation. This can be an expensive database operation, so Type 2 SCDs are not a good choice if the dimensional model is subject to change.

Type 3: Add New Attribute

This method tracks changes using separate columns and preserves limited history. The Type 3 preserves limited history as it is limited to the number of columns designated for storing historical data. The original table structure in Type 1 and Type 2 is the same but Type 3 adds additional columns. In the following example, an additional column has been added to the table to record the supplier's original state - only the previous history is stored.

Supplier_ Key	Supplier_ Code	Supplier_ Name	Original_Supplier_State	Effective_ Date	Current_Supplier_ State
123	ABC	Acme Supply Co	CA	22-Dec-2004	IL

This record contains a column for the original state and current state—cannot track the changes if the supplier relocates a second time.

One variation of this is to create the field Previous_Supplier_State instead of Original_Supplier_State which would track only the most recent historical change.

Type 4: Add History Table

The Type 4 method is usually referred to as using "history tables", where one table keeps the current data, and an additional table is used to keep a record of some or all changes. Both the surrogate keys are referenced in the Fact table to enhance query performance.

For the above example the original table name is Supplier and the history table is Supplier_History.

Supplier			
Supplier_key	Supplier_Code	Supplier_Name	Supplier_State
124	ABC	Acme & Johnson Supply Co	IL

Supplier_History				
Supplier_key	Supplier_Code	Supplier_Name	Supplier_State	Create_Date
123	ABC	Acme Supply Co	CA	14-June-2003
124	ABC	Acme & Johnson Supply Co	IL	22-Dec-2004

This method resembles how database audit tables and change data capture techniques function.

Type 6: Hybrid

The Type 6 method combines the approaches of types 1, 2 and 3 (1 + 2 + 3 = 6). One possible explanation of the origin of the term was that it was coined by Ralph Kimball during a conversation with Stephen Pace from Kalido. Ralph Kimball calls this method "Unpredictable Changes with Single-Version Overlay" in *The Data Warehouse Toolkit*.

The Supplier table starts out with one record for our example supplier:

Supplier_ Key	Row_ Key	Supplier_ Code	Supplier_ Name	Current_ State	Historical_ State	Start_ Date	End_ Date	Current_ Flag
123	1	ABC	Acme Supply Co	CA	CA	01-Jan-2000	31-Dec-2009	Y

The Current_State and the Historical_State are the same. The optional Current_Flag attribute indicates that this is the current or most recent record for this supplier.

When Acme Supply Company moves to Illinois, we add a new record, as in Type 2 processing, however a row key is included to ensure we have a unique key for each row:

Supplier_ Key	Row_ Key	Supplier_ Code	Supplier_ Name	Current_ State	Historical_ State	Start_ Date	End_ Date	Current_ Flag
123	1	ABC	Acme Supply Co	IL	CA	01-Jan-2000	21-Dec-2004	N
123	2	ABC	Acme Supply Co	IL	IL	22-Dec-2004	31-Dec-2009	Y

We overwrite the Current_Flag information in the first record (Row_Key = 1) with the new information, as in Type 1 processing. We create a new record to track the changes, as in Type 2 processing. And we store the history in a second State column (Historical_State), which incorporates Type 3 processing.

For example, if the supplier were to relocate again, we would add another record to the Supplier dimension, and we would overwrite the contents of the Current_State column:

Supplier_ Key	Row_ Key	Supplier_ Code	Supplier_ Name	Current_ State	Historical_ State	Start_ Date	End_ Date	Current_ Flag
123	1	ABC	Acme Supply Co	NY	CA	01-Jan-2000	21-Dec-2004	N
123	2	ABC	Acme Supply Co	NY	IL	22-Dec-2004	03-Feb-2008	N
123	3	ABC	Acme Supply Co	NY	NY	04-Feb-2008	31-Dec-2009	Y

Note that, for the current record (Current_Flag = 'Y'), the Current_State and the Historical_State are always the same.

Type 2 / type 6 Fact Implementation

Type 2 Surrogate Key with Type 3 Attribute

In many Type 2 and Type 6 SCD implementations, the surrogate key from the dimension is put into the fact table in place of the natural key when the fact data is loaded into the data repository. The surrogate key is selected for a given fact record based on its effective date and the Start_Date and End_Date from the dimension table. This allows the fact data to be easily joined to the correct dimension data for the corresponding effective date.

Here is the Supplier table as we created it above using Type 6 Hybrid methodology:

Supplier_ Key	Supplier_ Code	Supplier_ Name	Current_ State	Historical_ State	Start_ Date	End_ Date	Current_ Flag
123	ABC	Acme Supply Co	NY	CA	01-Jan-2000	21-Dec-2004	N
124	ABC	Acme Supply Co	NY	IL	22-Dec-2004	03-Feb-2008	N
125	ABC	Acme Supply Co	NY	NY	04-Feb-2008	31-Dec-9999	Y

Once the Delivery table contains the correct Supplier_Key, it can easily be joined to the Supplier table using that key. The following SQL retrieves, for each fact record, the current supplier state and the state the supplier was located in at the time of the delivery:

```
SELECT
  delivery.delivery_cost,
  supplier.supplier_name,
  supplier.historical_state,
  supplier.current_state
FROM delivery
INNER JOIN supplier
  ON delivery.supplier_key = supplier.supplier_key
```

Pure type 6 Implementation

Having a Type 2 surrogate key for each time slice can cause problems if the dimension is subject to change.

A pure Type 6 implementation does not use this, but uses a Surrogate Key for each master data item (e.g. each unique supplier has a single surrogate key).

This avoids any changes in the master data having an impact on the existing transaction data.

It also allows more options when querying the transactions.

Here is the Supplier table using the pure Type 6 methodology:

Supplier_Key	Supplier_Code	Supplier_Name	Supplier_State	Start_Date	End_Date
456	ABC	Acme Supply Co	CA	01-Jan-2000	21-Dec-2004
456	ABC	Acme Supply Co	IL	22-Dec-2004	03-Feb-2008
456	ABC	Acme Supply Co	NY	04-Feb-2008	31-Dec-9999

The following example shows how the query must be extended to ensure a single supplier record is retrieved for each transaction.

```
SELECT
 supplier.supplier_code,
 supplier.supplier_state
FROM supplier
INNER JOIN delivery
 ON supplier.supplier_key = delivery.supplier_key
AND delivery.delivery_date BETWEEN supplier.start_date AND supplier.end_date
```

A fact record with an effective date (Delivery_Date) of August 9, 2001 will be linked to Supplier_Code of ABC, with a Supplier_State of 'CA'. A fact record with an effective date of October 11, 2007 will also be linked to the same Supplier_Code ABC, but with a Supplier_State of 'IL'.

Whilst more complex, there are a number of advantages of this approach, including:

1. Referential integrity by DBMS is now possible, but one cannot use Supplier_Code as foreign key on Product table and using Supplier_Key as foreign key each product is tied on specific time slice.
2. If there is more than one date on the fact (e.g. Order Date, Delivery Date, Invoice Payment Date) one can choose which date to use for a query.
3. You can do "as at now", "as at transaction time" or "as at a point in time" queries by changing the date filter logic.
4. You don't need to reprocess the Fact table if there is a change in the dimension table (e.g. adding additional fields retrospectively which change the time slices, or if one makes a mistake in the dates on the dimension table one can correct them easily).
5. You can introduce bi-temporal dates in the dimension table.
6. You can join the fact to the multiple versions of the dimension table to allow reporting of the same information with different effective dates, in the same query.

The following example shows how a specific date such as '2012-01-01 00:00:00' (which could be the current datetime) can be used.

```
SELECT
 supplier.supplier_code,
 supplier.supplier_state
FROM supplier
INNER JOIN delivery
 ON supplier.supplier_key = delivery.supplier_key
AND '2012-01-01 00:00:00' BETWEEN supplier.start_date AND supplier.end_date
```

Both Surrogate and Natural Key

An alternative implementation is to place *both* the surrogate key and the natural key into the fact table. This allows the user to select the appropriate dimension records based on:

- the primary effective date on the fact record (above),
- the most recent or current information,
- any other date associated with the fact record.

This method allows more flexible links to the dimension, even if one has used the Type 2 approach instead of Type 6.

Here is the Supplier table as we might have created it using Type 2 methodology:

Supplier_Key	Supplier_Code	Supplier_Name	Supplier_State	Start_Date	End_Date	Current_Flag
123	ABC	Acme Supply Co	CA	01-Jan-2000	21-Dec-2004	N
124	ABC	Acme Supply Co	IL	22-Dec-2004	03-Feb-2008	N
125	ABC	Acme Supply Co	NY	04-Feb-2008	31-Dec-9999	Y

The following SQL retrieves the most current Supplier_Name and Supplier_State for each fact record:

```
SELECT
 delivery.delivery_cost,
 supplier.supplier_name,
 supplier.supplier_state
FROM delivery
```

```
INNER JOIN supplier
 ON delivery.supplier_code = supplier.supplier_code
WHERE supplier.current_flag = 'Y'
```

If there are multiple dates on the fact record, the fact can be joined to the dimension using another date instead of the primary effective date. For instance, the Delivery table might have a primary effective date of Delivery_Date, but might also have an Order_Date associated with each record.

The following SQL retrieves the correct Supplier_Name and Supplier_State for each fact record based on the Order_Date:

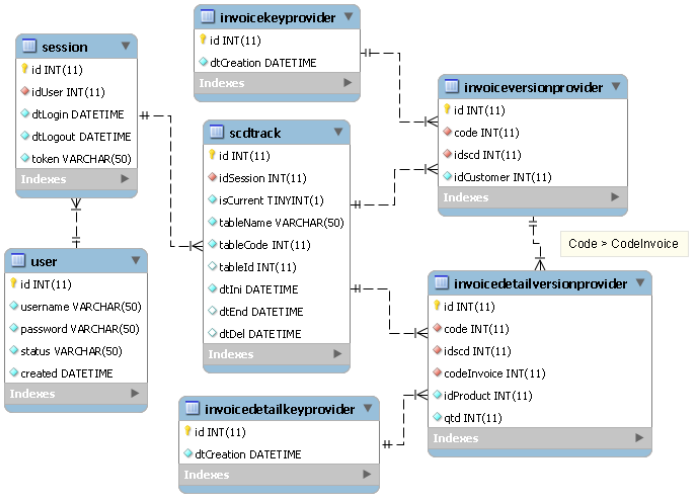
```
SELECT
 delivery.delivery_cost,
 supplier.supplier_name,
 supplier.supplier_state
FROM delivery
INNER JOIN supplier
 ON delivery.supplier_code = supplier.supplier_code
 AND delivery.order_date BETWEEN supplier.start_date AND supplier.end_date
```

Some cautions:

- Referential integrity by DBMS is not possible since there is not a unique to create the relationship.
- If relationship is made with surrogate to solve problem above then one ends with entity tied to a specific time slice.
- If the join query is not written correctly, it may return duplicate rows and/or give incorrect answers.
- The date comparison might not perform well.
- Some Business Intelligence tools do not handle generating complex joins well.
- The ETL processes needed to create the dimension table needs to be carefully designed to ensure that there are no overlaps in the time periods for each distinct item of reference data.
- Many of problems above can be solved using the mixed diagram of an scd model below.

Combining Types

Different SCD Types can be applied to different columns of a table. For example, we can apply Type 1 to the Supplier_Name column and Type 2 to the Supplier_State column of the same table.



Scd model

Data Vault Modeling

Data vault modeling is a database modeling method that is designed to provide long-term historical storage of data coming in from multiple operational systems. It is also a method of looking at historical data that deals with issues such as auditing, tracing of data, loading speed and resilience to change as well as emphasizing the need to trace where all the data in the database came from. This means that every row in a data vault must be accompanied by record source and load date attributes, enabling an auditor to trace values back to the source.

Data vault modeling makes no distinction between good and bad data ("bad" meaning not conforming to business rules). This is summarized in the statement that a data vault stores "a single version of the facts" (also expressed by Dan Linstedt as "all the data, all of the time") as opposed to the practice in other data warehouse methods of storing "a single version of the truth" where data that does not conform to the definitions is removed or "cleansed".

The modeling method is designed to be resilient to change in the business environment where the data being stored is coming from, by explicitly separating structural information from descriptive attributes. Data vault is designed to enable parallel loading as much as possible, so that very large implementations can scale out without the need for major redesign.

History and Philosophy

In data warehouse modeling there are two well-known competing options for modeling the layer where the data are stored. Either you model according to Ralph Kimball, with conformed dimensions and an enterprise data bus, or you model according to Bill Inmon with the database normalized. Both techniques have issues when dealing with changes in the systems feeding the data warehouse. For conformed dimensions you also have to cleanse data (to conform it) and this is undesirable in a number of cases since this inevitably will lose information. Data vault is designed to avoid or minimize the impact of those issues, by moving them to areas of the data warehouse

that are outside the historical storage area (cleansing is done in the data marts) and by separating the structural items (business keys and the associations between the business keys) from the descriptive attributes.

Dan Linstedt, the creator of the method, describes the resulting database as follows:

The Data Vault Model is a detail oriented, historical tracking and uniquely linked set of normalized tables that support one or more functional areas of business. It is a hybrid approach encompassing the best of breed between 3rd normal form (3NF) and star schema. The design is flexible, scalable, consistent and adaptable to the needs of the enterprise

Data vault's philosophy is that all data are relevant data, even if it is not in line with established definitions and business rules. If data are not conforming to these definitions and rules then that is a problem for the business, not the data warehouse. The determination of data being "wrong" is an interpretation of the data that stems from a particular point of view that may not be valid for everyone, or at every point in time. Therefore the data vault must capture all data and only when reporting or extracting data from the data vault is the data being interpreted.

Another issue to which data vault is a response is that more and more there is a need for complete auditability and traceability of all the data in the data warehouse. Due to Sarbanes-Oxley requirements in the USA and similar measures in Europe this is a relevant topic for many business intelligence implementations, hence the focus of any data vault implementation is complete traceability and auditability of all information.

Data Vault 2.0 is the new specification, it is an open standard. The new specification contains components which define the implementation best practices, the methodology (SEI/CMMI, Six Sigma, SDLC, etc..), the architecture, and the model. *Data Vault 2.0* has a focus on including new components such as Big Data, NoSQL - and also focuses on performance of the existing model. The old specification (documented here for the most part) is highly focused on data vault modeling. It is documented in the book: Building a Scalable Data Warehouse with Data Vault 2.0.

It is necessary to evolve the specification to include the new components, along with the best practices in order to keep the EDW and BI systems current with the needs and desires of today's businesses.

History

Data vault modeling was originally conceived by Dan Linstedt in 1990 and was released in 2000 as a public domain modeling method. In a series of five articles on The Data Administration Newsletter the basic rules of the Data Vault method are expanded and explained. These contain a general overview, an overview of the components, a discussion about end dates and joins, link tables, and an article on loading practices.

An alternative (and seldom used) name for the method is "Common Foundational Integration Modelling Architecture."

Data Vault 2.0 has arrived on the scene as of 2013 and brings to the table Big Data, NoSQL, unstructured, semi-structured seamless integration, along with methodology, architecture, and implementation best practices.

Alternative Interpretations

According to Dan Linstedt, the Data Model is inspired by (or patterned off) a simplistic view of neurons, dendrites, and synapses – where neurons are associated with Hubs and Hub Satellites, Links are dendrites (vectors of information), and other Links are synapses (vectors in the opposite direction). By using a data mining set of algorithms, links can be scored with confidence and strength ratings. They can be created and dropped on the fly in accordance with learning about relationships that currently don't exist. The model can be automatically morphed, adapted, and adjusted as it is used and fed new structures.

Another view is that a data vault model provides an ontology of the Enterprise in the sense that it describes the terms in the domain of the enterprise (Hubs) and the relationships among them (Links), adding descriptive attributes (Satellites) where necessary.

Another way to think of a data vault model is as a graph model. The data vault model actually provides a "graph based" model with hubs and relationships in a relational database world. In this manner, the developer can use SQL to get at graph based relationships with sub-second responses.

Basic Notions

Data vault attempts to solve the problem of dealing with change in the environment by separating the business keys (that do not mutate as often, because they uniquely identify a business entity) and the associations between those business keys, from the descriptive attributes of those keys.

The business keys and their associations are structural attributes, forming the skeleton of the data model. The data vault method has as one of its main axioms that real business keys only change when the business changes and are therefore the most stable elements from which to derive the structure of a historical database. If you use these keys as the backbone of a data warehouse, you can organize the rest of the data around them. This means that choosing the correct keys for the hubs is of prime importance for the stability of your model. The keys are stored in tables with a few constraints on the structure. These key-tables are called hubs.

Hubs

Hubs contain a list of unique business keys with low propensity to change. Hubs also contain a surrogate key for each Hub item and metadata describing the origin of the business key. The descriptive attributes for the information on the Hub (such as the description for the key, possibly in multiple languages) are stored in structures called Satellite tables which will be discussed below.

The Hub contains at least the following fields:

- a surrogate key, used to connect the other structures to this table.
- a business key, the driver for this hub. The business key can consist of multiple fields.
- the record source, which can be used to see what system loaded each business key first.

optionally, you can also have metadata fields with information about manual updates (user/time) and the extraction date.

A hub is not allowed to contain multiple business keys, except when two systems deliver the same business key but with collisions that have different meanings.

Hubs should normally have at least one satellite.

Hub Example

This is an example for a hub-table containing cars, called "Car" (H_CAR). The driving key is vehicle identification number.

Fieldname	Description	Mandatory?	Comment
H_CAR_ID	Sequence ID and surrogate key for the hub	No	Recommended but optional
VEHICLE_ID_NR	The business key that drives this hub. Can be more than one field for a composite business key	Yes	
H_RSRC	The recordsource of this key when first loaded	Yes	
LOAD_AUDIT_ID	An ID into a table with audit information, such as load time, duration of load, number of lines, etc.	No	

Links

Associations or transactions between business keys (relating for instance the hubs for customer and product with each other through the purchase transaction) are modeled using link tables. These tables are basically many-to-many join tables, with some metadata.

Links can link to other links, to deal with changes in granularity (for instance, adding a new key to a database table would change the grain of the database table). For instance, if you have an association between customer and address, you could add a reference to a link between the hubs for product and transport company. This could be a link called "Delivery". Referencing a link in another link is considered a bad practice, because it introduces dependencies between links that make parallel loading more difficult. Since a link to another link is the same as a new link with the hubs from the other link, in these cases creating the links without referencing other links is the preferred solution.

Links sometimes link hubs to information that is not by itself enough to construct a hub. This occurs when one of the business keys associated by the link is not a real business key. As an example, take an order form with "order number" as key, and order lines that are keyed with a semi-random number to make them unique. Let's say, "unique number". The latter key is not a real business key, so it is no hub. However, we do need to use it in order to guarantee the correct granularity for the link. In this case, we do not use a hub with surrogate key, but add the business key "unique number" itself to the link. This is done only when there is no possibility of ever using the business key for another link or as key for attributes in a satellite. This construct has been called a 'peg-legged link' by Dan Linstedt on his (now defunct) forum.

Links contain the surrogate keys for the hubs that are linked, their own surrogate key for the link and metadata describing the origin of the association. The descriptive attributes for the information on the association (such as the time, price or amount) are stored in structures called *satellite tables* which are discussed below.

Link Example

This is an example for a link-table between two hubs for cars (H_CAR) and persons (H_PERSON). The link is called "Driver" (L_DRIVER).

Fieldname	Description	Mandatory?	Comment
L_DRIVER_ID	Sequence ID and surrogate key for the Link	No	Recommended but optional
H_CAR_ID	surrogate key for the car hub, the first anchor of the link	Yes	
H_PERSON_ID	surrogate key for the person hub, the second anchor of the link	Yes	
L_RSRC	The recordsource of this association when first loaded	Yes	
LOAD_AUDIT_ID	An ID into a table with audit information, such as load time, duration of load, number of lines, etc.	No	

Satellites

The hubs and links form the structure of the model, but have no temporal attributes and hold no descriptive attributes. These are stored in separate tables called *satellites*. These consist of meta-data linking them to their parent hub or link, metadata describing the origin of the association and attributes, as well as a timeline with start and end dates for the attribute. Where the hubs and links provide the structure of the model, the satellites provide the "meat" of the model, the context for the business processes that are captured in hubs and links. These attributes are stored both with regards to the details of the matter as well as the timeline and can range from quite complex (all of the fields describing a clients complete profile) to quite simple (a satellite on a link with only a valid-indicator and a timeline).

Usually the attributes are grouped in satellites by source system. However, descriptive attributes such as size, cost, speed, amount or color can change at different rates, so you can also split these attributes up in different satellites based on their rate of change.

All the tables contain metadata, minimally describing at least the source system and the date on which this entry became valid, giving a complete historical view of the data as it enters the data warehouse.

Satellite Example

This is an example for a satellite on the drivers-link between the hubs for cars and persons, called "Driver insurance" (S_DRIVER_INSURANCE). This satellite contains attributes that are specific to the insurance of the relationship between the car and the person driving it, for instance an indicator whether this is the primary driver, the name of the insurance company for this car and person (could also be a separate hub) and a summary of the number of accidents involving this combination of vehicle and driver. Also included is a reference to a lookup- or reference table called R_RISK_CATEGORY containing the codes for the risk category in which this relationship is deemed to fall.

Fieldname	Description	Mandatory?	Comment
S_DRIVER_INSURANCE_ID	Sequence ID and surrogate key for the satellite on the link	No	Recommended but optional
L_DRIVER_ID	(surrogate) primary key for the driver link, the parent of the satellite	Yes	
S_SEQ_NR	Ordering or sequence number, to enforce uniqueness if there are several valid satellites for one parent key	No(**)	This can happen if, for instance, you have a hub COURSE and the name of the course is an attribute but in several different languages.
S_LDTS	Load Date (startdate) for the validity of this combination of attribute values for parent key L_DRIVER_ID	Yes	
S_LEDTS	Load End Date (enddate) for the validity of this combination of attribute values for parent key L_DRIVER_ID	No	
IND_PRIMARY_DRIVER	Indicator whether the driver is the primary driver for this car	No (*)	
INSURANCE_COMPANY	The name of the insurance company for this vehicle and this driver	No (*)	
NR_OF_ACCIDENTS	The number of accidents by this driver in this vehicle	No (*)	
R_RISK_CATEGORY_CD	The risk category for the driver. This is a reference to R_RISK_CATEGORY	No (*)	
S_RSRC	The recordsource of the information in this satellite when first loaded	Yes	
LOAD_AUDIT_ID	An ID into a table with audit information, such as load time, duration of load, number of lines, etc.	No	

(*) at least one attribute is mandatory. (**) sequence number becomes mandatory if it is needed to enforce uniqueness for multiple valid satellites on the same hub or link.

Reference Tables

Reference tables are a normal part of a healthy data vault model. They are there to prevent redundant storage of simple reference data that is referenced a lot. More formally, Dan Linstedt defines reference data as follows:

Any information deemed necessary to resolve descriptions from codes, or to translate keys in to (sic) a consistent manner. Many of these fields are "descriptive" in nature and ***describe*** *a specific state of the other more important information. As such, reference data lives in separate tables from the raw Data Vault tables.*

Reference tables are referenced from Satellites, but never bound with physical foreign keys. There is no prescribed structure for reference tables: use what works best in your specific case, ranging from simple lookup tables to small data vaults or even stars. They can be historical or have no history, but it is recommended that you stick to the natural keys and not create surrogate keys in that case. Normally, data vaults have a lot of reference tables, just like any other Data Warehouse.

Reference Example

This is an example of a reference table with risk categories for drivers of vehicles. It can be referenced from any satellite in the data vault. For now we reference it from satellite S_DRIVER_INSURANCE. The reference table is R_RISK_CATEGORY.

Fieldname	Description	Mandatory?
R_RISK_CATEGORY_CD	The code for the risk category	Yes
RISK_CATEGORY_DESC	A description of the risk category	No (*)

(*) at least one attribute is mandatory.

Loading Practices

The ETL for updating a data vault model is fairly straightforward. First you have to load all the hubs, creating surrogate IDs for any new business keys. Having done that, you can now resolve all business keys to surrogate ID's if you query the hub. The second step is to resolve the links between hubs and create surrogate IDs for any new associations. At the same time, you can also create all satellites that are attached to hubs, since you can resolve the key to a surrogate ID. Once you have created all the new links with their surrogate keys, you can add the satellites to all the links.

Since the hubs are not joined to each other except through links, you can load all the hubs in parallel. Since links are not attached directly to each other, you can load all the links in parallel as well. Since satellites can be attached only to hubs and links, you can also load these in parallel.

The ETL is quite straightforward and lends itself to easy automation or templating. Problems occur only with links relating to other links, because resolving the business keys in the link only leads to another link that has to be resolved as well. Due to the equivalence of this situation with a link to multiple hubs, this difficulty can be avoided by remodeling such cases and this is in fact the recommended practice.

Data are never deleted from the data vault, unless you have a technical error while loading data.

Data Vault and Dimensional Modelling

The data vault modelled layer is normally used to store data. It is not optimized for query performance, nor is it easy to query by the well-known query-tools such as Cognos, SAP Business Objects, Pentaho et al. Since these end-user computing tools expect or prefer their data to be contained in a dimensional model, a conversion is usually necessary.

For this purpose, the hubs and related satellites on those hubs can be considered as dimensions

and the links and related satellites on those links can be viewed as fact tables in a dimensional model. This enables you to quickly prototype a dimensional model out of a data vault model using views. For performance reasons the dimensional model will usually be implemented in relational tables, after approval.

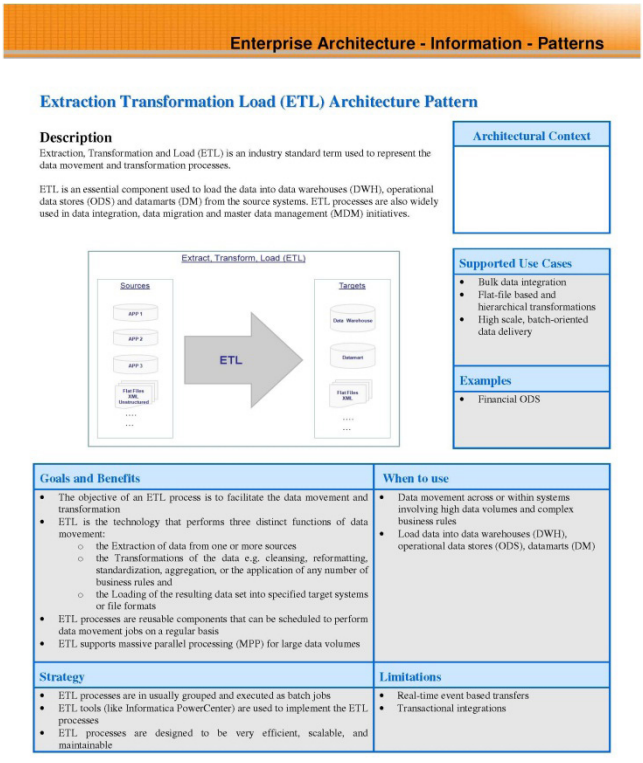
Note that while it is relatively straightforward to move data from a data vault model to a (cleansed) dimensional model, the reverse is not as easy.

Data Vault Methodology

The data vault methodology is based on SEI/CMMI Level 5 best practices. It includes multiple components of CMMI Level 5, and combines them with best practices from Six Sigma, TQM, and SDLC. Particularly, it is focused on Scott Ambler's agile methodology for build out and deployment. Data vault projects have a short, scope-controlled release cycle and should consist of a production release every 2 to 3 weeks.

Teams using the data vault methodology will automatically adopt to the repeatable, consistent, and measurable projects that are expected at CMMI Level 5. Data that flow through the EDW data vault system will begin to follow the TQM (total quality management) life-cycle that has long been missing from BI (business intelligence) projects.

Extract, Transform, Load



ETL Architecture Pattern

In computing, Extract, Transform, Load (ETL) refers to a process in database usage and especially in data warehousing. Data extraction is where data is extracted from homogeneous or heterogeneous data sources; data transformation where the data is transformed for storing in the proper format or structure for the purposes of querying and analysis; data loading where the data is loaded into the final target database, more specifically, an operational data store, data mart, or data warehouse.

Since the data extraction takes time, it is common to execute the three phases in parallel. While the data is being extracted, another transformation process executes. It processes the already received data and prepares it for loading. As soon as there is some data ready to be loaded into the target, the data loading kicks off without waiting for the completion of the previous phases.

ETL systems commonly integrate data from multiple applications (systems), typically developed and supported by different vendors or hosted on separate computer hardware. The disparate systems containing the original data are frequently managed and operated by different employees. For example, a cost accounting system may combine data from payroll, sales, and purchasing.

Extract

The first part of an ETL process involves extracting the data from the source system(s). In many cases this represents the most important aspect of ETL, since extracting data correctly sets the stage for the success of subsequent processes. Most data-warehousing projects combine data from different source systems. Each separate system may also use a different data organization and/or format. Common data-source formats include relational databases, XML and flat files, but may also include non-relational database structures such as Information Management System (IMS) or other data structures such as Virtual Storage Access Method (VSAM) or Indexed Sequential Access Method (ISAM), or even formats fetched from outside sources by means such as web spidering or screen-scraping. The streaming of the extracted data source and loading on-the-fly to the destination database is another way of performing ETL when no intermediate data storage is required. In general, the extraction phase aims to convert the data into a single format appropriate for transformation processing.

An intrinsic part of the extraction involves data validation to confirm whether the data pulled from the sources has the correct/expected values in a given domain (such as a pattern/default or list of values). If the data fails the validation rules it is rejected entirely or in part. The rejected data is ideally reported back to the source system for further analysis to identify and to rectify the incorrect records. In some cases the extraction process itself may have to do a data-validation rule in order to accept the data and flow to the next phase.

Transform

In the data transformation stage, a series of rules or functions are applied to the extracted data in order to prepare it for loading into the end target. Some data does not require any transformation at all; such data is known as “direct move” or “pass through” data.

An important function of transformation is the cleaning of data, which aims to pass only “proper” data to the target. The challenge when different systems interact is in the relevant systems’ inter-

facing and communicating. Character sets that may be available in one system may not be so in others.

In other cases, one or more of the following transformation types may be required to meet the business and technical needs of the server or data warehouse:

- Selecting only certain columns to load: (or selecting null columns not to load). For example, if the source data has three columns (aka "attributes"), roll_no, age, and salary, then the selection may take only roll_no and salary. Or, the selection mechanism may ignore all those records where salary is not present (salary = null).
- Translating coded values: (*e.g.*, if the source system codes male as "1" and female as "2", but the warehouse codes male as "M" and female as "F")
- Encoding free-form values: (*e.g.*, mapping "Male" to "M")
- Deriving a new calculated value: (*e.g.*, sale_amount = qty * unit_price)
- Sorting or ordering the data based on a list of columns to improve search performance
- Joining data from multiple sources (*e.g.*, lookup, merge) and deduplicating the data
- Aggregating (for example, rollup — summarizing multiple rows of data — total sales for each store, and for each region, etc.)
- Generating surrogate-key values
- Transposing or pivoting (turning multiple columns into multiple rows or vice versa)
- Splitting a column into multiple columns (*e.g.*, converting a comma-separated list, specified as a string in one column, into individual values in different columns)
- Disaggregating repeating columns
- Looking up and validating the relevant data from tables or referential files
- Applying any form of data validation; failed validation may result in a full rejection of the data, partial rejection, or no rejection at all, and thus none, some, or all of the data is handed over to the next step depending on the rule design and exception handling; many of the above transformations may result in exceptions, e.g., when a code translation parses an unknown code in the extracted data

Load

The load phase loads the data into the end target that may be a simple delimited flat file or a data warehouse. Depending on the requirements of the organization, this process varies widely. Some data warehouses may overwrite existing information with cumulative information; updating extracted data is frequently done on a daily, weekly, or monthly basis. Other data warehouses (or even other parts of the same data warehouse) may add new data in a historical form at regular intervals—for example, hourly. To understand this, consider a data warehouse that is required to maintain sales records of the last year. This data warehouse overwrites any data older than a year with newer data. However, the entry of data for any one year window is made in a historical man-

ner. The timing and scope to replace or append are strategic design choices dependent on the time available and the business needs. More complex systems can maintain a history and audit trail of all changes to the data loaded in the data warehouse.

As the load phase interacts with a database, the constraints defined in the database schema — as well as in triggers activated upon data load — apply (for example, uniqueness, referential integrity, mandatory fields), which also contribute to the overall data quality performance of the ETL process.

- For example, a financial institution might have information on a customer in several departments and each department might have that customer's information listed in a different way. The membership department might list the customer by name, whereas the accounting department might list the customer by number. ETL can bundle all of these data elements and consolidate them into a uniform presentation, such as for storing in a database or data warehouse.
- Another way that companies use ETL is to move information to another application permanently. For instance, the new application might use another database vendor and most likely a very different database schema. ETL can be used to transform the data into a format suitable for the new application to use.
- An example would be an Expense and Cost Recovery System (ECRS) such as used by accountancies, consultancies, and legal firms. The data usually ends up in the time and billing system, although some businesses may also utilize the raw data for employee productivity reports to Human Resources (personnel dept.) or equipment usage reports to Facilities Management.

Real-life ETL Cycle

The typical real-life ETL cycle consists of the following execution steps:

1. Cycle initiation
2. Build reference data
3. Extract (from sources)
4. Validate
5. Transform (clean, apply business rules, check for data integrity, create aggregates or disaggregates)
6. Stage (load into staging tables, if used)
7. Audit reports (for example, on compliance with business rules. Also, in case of failure, helps to diagnose/repair)
8. Publish (to target tables)
9. Archive

Challenges

ETL processes can involve considerable complexity, and significant operational problems can oc-

cur with improperly designed ETL systems.

The range of data values or data quality in an operational system may exceed the expectations of designers at the time validation and transformation rules are specified. Data profiling of a source during data analysis can identify the data conditions that must be managed by transform rules specifications, leading to an amendment of validation rules explicitly and implicitly implemented in the ETL process.

Data warehouses are typically assembled from a variety of data sources with different formats and purposes. As such, ETL is a key process to bring all the data together in a standard, homogeneous environment.

Design analysts should establish the scalability of an ETL system across the lifetime of its usage---including understanding the volumes of data that must be processed within service level agreements. The time available to extract from source systems may change, which may mean the same amount of data may have to be processed in less time. Some ETL systems have to scale to process terabytes of data to update data warehouses with tens of terabytes of data. Increasing volumes of data may require designs that can scale from daily batch to multiple-day micro batch to integration with message queues or real-time change-data capture for continuous transformation and update.

Performance

ETL vendors benchmark their record-systems at multiple TB (terabytes) per hour (or ~1 GB per second) using powerful servers with multiple CPUs, multiple hard drives, multiple gigabit-network connections, and lots of memory. The fastest ETL record is currently held by Syncsort, Vertica, and HP at 5.4TB in under an hour, which is more than twice as fast as the earlier record held by Microsoft and Unisys.

In real life, the slowest part of an ETL process usually occurs in the database load phase. Databases may perform slowly because they have to take care of concurrency, integrity maintenance, and indices. Thus, for better performance, it may make sense to employ:

- Direct Path Extract method or bulk unload whenever is possible (instead of querying the database) to reduce the load on source system while getting high speed extract
- Most of the transformation processing outside of the database
- Bulk load operations whenever possible

Still, even using bulk operations, database access is usually the bottleneck in the ETL process. Some common methods used to increase performance are:

- Partition tables (and indices): try to keep partitions similar in size (watch for null values that can skew the partitioning)
- Do all validation in the ETL layer before the load: disable integrity checking (disable constraint ...) in the target database tables during the load
- Disable triggers (disable trigger ...) in the target database tables during the load: simulate their effect as a separate step

- Generate IDs in the ETL layer (not in the database)
- Drop the indices (on a table or partition) before the load - and recreate them after the load (SQL: drop index ...; create index ...)
- Use parallel bulk load when possible — works well when the table is partitioned or there are no indices (Note: attempt to do parallel loads into the same table (partition) usually causes locks — if not on the data rows, then on indices)
- If a requirement exists to do insertions, updates, or deletions, find out which rows should be processed in which way in the ETL layer, and then process these three operations in the database separately; you often can do bulk load for inserts, but updates and deletes commonly go through an API (using SQL)

Whether to do certain operations in the database or outside may involve a trade-off. For example, removing duplicates using distinct may be slow in the database; thus, it makes sense to do it outside. On the other side, if using distinct significantly (x100) decreases the number of rows to be extracted, then it makes sense to remove duplications as early as possible in the database before unloading data.

A common source of problems in ETL is a big number of dependencies among ETL jobs. For example, job "B" cannot start while job "A" is not finished. One can usually achieve better performance by visualizing all processes on a graph, and trying to reduce the graph making maximum use of parallelism, and making "chains" of consecutive processing as short as possible. Again, partitioning of big tables and their indices can really help.

Another common issue occurs when the data are spread among several databases, and processing is done in those databases sequentially. Sometimes database replication may be involved as a method of copying data between databases - it can significantly slow down the whole process. The common solution is to reduce the processing graph to only three layers:

- Sources
- Central ETL layer
- Targets

This approach allows processing to take maximum advantage of parallelism. For example, if you need to load data into two databases, you can run the loads in parallel (instead of loading into the first - and then replicating into the second).

Sometimes processing must take place sequentially. For example, dimensional (reference) data are needed before one can get and validate the rows for main "fact" tables.

Parallel Processing

A recent development in ETL software is the implementation of parallel processing. It has enabled a number of methods to improve overall performance of ETL when dealing with large volumes of data.

ETL applications implement three main types of parallelism:

- Data: By splitting a single sequential file into smaller data files to provide parallel access
- Pipeline: allowing the simultaneous running of several components on the same data stream, e.g. looking up a value on record 1 at the same time as adding two fields on record 2
- Component: The simultaneous running of multiple processes on different data streams in the same job, e.g. sorting one input file while removing duplicates on another file

All three types of parallelism usually operate combined in a single job.

An additional difficulty comes with making sure that the data being uploaded is relatively consistent. Because multiple source databases may have different update cycles (some may be updated every few minutes, while others may take days or weeks), an ETL system may be required to hold back certain data until all sources are synchronized. Likewise, where a warehouse may have to be reconciled to the contents in a source system or with the general ledger, establishing synchronization and reconciliation points becomes necessary.

Rerunnability, Recoverability

Data warehousing procedures usually subdivide a big ETL process into smaller pieces running sequentially or in parallel. To keep track of data flows, it makes sense to tag each data row with "row_id", and tag each piece of the process with "run_id". In case of a failure, having these IDs help to roll back and rerun the failed piece.

Best practice also calls for *checkpoints*, which are states when certain phases of the process are completed. Once at a checkpoint, it is a good idea to write everything to disk, clean out some temporary files, log the state, and so on.

Virtual ETL

As of 2010 data virtualization had begun to advance ETL processing. The application of data virtualization to ETL allowed solving the most common ETL tasks of data migration and application integration for multiple dispersed data sources. Virtual ETL operates with the abstracted representation of the objects or entities gathered from the variety of relational, semi-structured, and unstructured data sources. ETL tools can leverage object-oriented modeling and work with entities' representations persistently stored in a centrally located hub-and-spoke architecture. Such a collection that contains representations of the entities or objects gathered from the data sources for ETL processing is called a metadata repository and it can reside in memory or be made persistent. By using a persistent metadata repository, ETL tools can transition from one-time projects to persistent middleware, performing data harmonization and data profiling consistently and in near-real time.

Dealing with Keys

Keys play an important part in all relational databases, as they tie everything together. A primary key is a column that identifies a given entity, whereas a foreign key is a column in another table that refers to a primary key. Keys can comprise several columns, in which case they are composite keys. In many cases the primary key is an auto-generated integer that has no meaning for the busi-

ness entity being represented, but solely exists for the purpose of the relational database - commonly referred to as a surrogate key.

As there is usually more than one data source getting loaded into the warehouse, the keys are an important concern to be addressed. For example: customers might be represented in several data sources, with their Social Security Number as the primary key in one source, their phone number in another, and a surrogate in the third. Yet a data warehouse may require the consolidation of all the customer information into one dimension table.

A recommended way to deal with the concern involves adding a warehouse surrogate key, which is used as a foreign key from the fact table.

Usually updates occur to a dimension's source data, which obviously must be reflected in the data warehouse.

If the primary key of the source data is required for reporting, the dimension already contains that piece of information for each row. If the source data uses a surrogate key, the warehouse must keep track of it even though it is never used in queries or reports; it is done by creating a lookup table that contains the warehouse surrogate key and the originating key. This way, the dimension is not polluted with surrogates from various source systems, while the ability to update is preserved.

The lookup table is used in different ways depending on the nature of the source data. There are 5 types to consider; three are included here:

Type 1

> The dimension row is simply updated to match the current state of the source system; the warehouse does not capture history; the lookup table is used to identify the dimension row to update or overwrite

Type 2

> A new dimension row is added with the new state of the source system; a new surrogate key is assigned; source key is no longer unique in the lookup table

Fully logged

> A new dimension row is added with the new state of the source system, while the previous dimension row is updated to reflect it is no longer active and time of deactivation.

Tools

By using an established ETL framework, one may increase one's chances of ending up with better connectivity and scalability. A good ETL tool must be able to communicate with the many different relational databases and read the various file formats used throughout an organization. ETL tools have started to migrate into Enterprise Application Integration, or even Enterprise Service Bus, systems that now cover much more than just the extraction, transformation, and loading of data. Many ETL vendors now have data profiling, data quality, and metadata capabilities. A common use case for ETL tools include converting CSV files to formats readable by relational databases. A typical translation of millions of records is facilitated by ETL tools that enable users to input csv-

like data feeds/files and import it into a database with as little code as possible.

ETL tools are typically used by a broad range of professionals - from students in computer science looking to quickly import large data sets to database architects in charge of company account management, ETL tools have become a convenient tool that can be relied on to get maximum performance. ETL tools in most cases contain a GUI that helps users conveniently transform data, using a visual data mapper, as opposed to writing large programs to parse files and modify data types.

While ETL Tools have traditionally been for developers and I.T. staff, the new trend is to provide these capabilities to business users so they can themselves create connections and data integrations when needed, rather than going to the I.T. staff. Gartner refers to these non-technical users as Citizen Integrators.

Star Schema

In computing, the star schema is the simplest style of data mart schema and is the approach most widely used to develop data warehouses and dimensional data marts. The star schema consists of one or more fact tables referencing any number of dimension tables. The star schema is an important special case of the snowflake schema, and is more effective for handling simpler queries.

The star schema gets its name from the physical model's resemblance to a star shape with a fact table at its center and the dimension tables surrounding it representing the star's points.

Model

The star schema separates business process data into facts, which hold the measurable, quantitative data about a business, and dimensions which are descriptive attributes related to fact data. Examples of fact data include sales price, sale quantity, and time, distance, speed, and weight measurements. Related dimension attribute examples include product models, product colors, product sizes, geographic locations, and salesperson names.

A star schema that has many dimensions is sometimes called a *centipede schema*. Having dimensions of only a few attributes, while simpler to maintain, results in queries with many table joins and makes the star schema less easy to use.

Fact Tables

Fact tables record measurements or metrics for a specific event. Fact tables generally consist of numeric values, and foreign keys to dimensional data where descriptive information is kept. Fact tables are designed to a low level of uniform detail (referred to as "granularity" or "grain"), meaning facts can record events at a very atomic level. This can result in the accumulation of a large number of records in a fact table over time. Fact tables are defined as one of three types:

- Transaction fact tables record facts about a specific event (e.g., sales events)
- Snapshot fact tables record facts at a given point in time (e.g., account details at month end)

- Accumulating snapshot tables record aggregate facts at a given point in time (e.g., total month-to-date sales for a product)

Fact tables are generally assigned a surrogate key to ensure each row can be uniquely identified. This key is a simple primary key.

Dimension Tables

Dimension tables usually have a relatively small number of records compared to fact tables, but each record may have a very large number of attributes to describe the fact data. Dimensions can define a wide variety of characteristics, but some of the most common attributes defined by dimension tables include:

- Time dimension tables describe time at the lowest level of time granularity for which events are recorded in the star schema
- Geography dimension tables describe location data, such as country, state, or city
- Product dimension tables describe products
- Employee dimension tables describe employees, such as sales people
- Range dimension tables describe ranges of time, dollar values, or other measurable quantities to simplify reporting

Dimension tables are generally assigned a surrogate primary key, usually a single-column integer data type, mapped to the combination of dimension attributes that form the natural key.

Benefits

Star schemas are denormalized, meaning the normal rules of normalization applied to transactional relational databases are relaxed during star schema design and implementation. The benefits of star schema denormalization are:

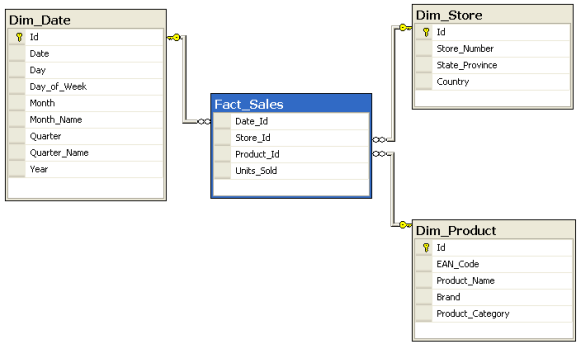
- Simpler queries - star schema join logic is generally simpler than the join logic required to retrieve data from a highly normalized transactional schema.
- Simplified business reporting logic - when compared to highly normalized schemas, the star schema simplifies common business reporting logic, such as period-over-period and as-of reporting.
- Query performance gains - star schemas can provide performance enhancements for read-only reporting applications when compared to highly normalized schemas.
- Fast aggregations - the simpler queries against a star schema can result in improved performance for aggregation operations.
- Feeding cubes - star schemas are used by all OLAP systems to build proprietary OLAP cubes efficiently; in fact, most major OLAP systems provide a ROLAP mode of operation which can use a star schema directly as a source without building a proprietary cube structure.

Disadvantages

The main disadvantage of the star schema is that data integrity is not enforced as well as it is in a highly normalized database. One-off inserts and updates can result in data anomalies which normalized schemas are designed to avoid. Generally speaking, star schemas are loaded in a highly controlled fashion via batch processing or near-real time "trickle feeds", to compensate for the lack of protection afforded by normalization.

Star schema is also not as flexible in terms of analytical needs as a normalized data model. Normalized models allow any kind of analytical queries to be executed as long as they follow the business logic defined in the model. Star schemas tend to be more purpose-built for a particular view of the data, thus not really allowing more complex analytics. Star schemas don't support many-to-many relationships between business entities - at least not very naturally. Typically these relationships are simplified in star schema to conform to the simple dimensional model.

Example



Star schema used by example query.

Consider a database of sales, perhaps from a store chain, classified by date, store and product. The image of the schema to the right is a star schema version of the sample schema provided in the snowflake schema article.

Fact_Sales is the fact table and there are three dimension tables Dim_Date, Dim_Store and Dim_Product.

Each dimension table has a primary key on its Id column, relating to one of the columns (viewed as rows in the example schema) of the Fact_Sales table's three-column (compound) primary key (Date_Id, Store_Id, Product_Id). The non-primary key Units_Sold column of the fact table in this example represents a measure or metric that can be used in calculations and analysis. The non-primary key columns of the dimension tables represent additional attributes of the dimensions (such as the Year of the Dim_Date dimension).

For example, the following query answers how many TV sets have been sold, for each brand and country, in 1997:

```
SELECT

        P.Brand,
```

```
        S.Country AS Countries,
        SUM(F.Units_Sold)
FROM Fact_Sales F
INNER JOIN Dim_Date D    ON (F.Date_Id = D.Id)
INNER JOIN Dim_Store S   ON (F.Store_Id = S.Id)
INNER JOIN Dim_Product P ON (F.Product_Id = P.Id)
WHERE D.Year = 1997 AND  P.Product_Category = 'tv'
GROUP BY
        P.Brand,
        S.Country
```

References

- Ralph Kimball, Margy Ross, The Data Warehouse Toolkit: The Complete Guide to Dimensional Modeling, Second Edition, Wiley Computer Publishing, 2002. ISBN 0471-20024-7.
- Ralph Kimball, The Data Warehouse Toolkit, Second Edition, Wiley Publishing, Inc., 2008. ISBN 978-0-470-14977-5, Pages 253-256
- Thomas C. Hammergren; Alan R. Simon (February 2009). Data Warehousing for Dummies, 2nd edition. John Wiley & Sons. ISBN 978-0-470-40747-9.
- "Information Theory & Business Intelligence Strategy - Small Worlds Data Transformation Measure - MIKE2.0, the open source methodology for Information Development". Mike2.openmethodology.org. Retrieved 2013-06-14.
- Linstedt, Dan. "Data Vault Series 1 – Data Vault Overview". Data Vault Series. The Data Administration Newsletter. Retrieved 12 September 2011.