EPAM Systems, RD Dep., RD Dep.

INTRODUCTION TO DWH AND ETL

Introduction to Data Warehousing

Legal Notice: This document contains privileged and/or confidential information and may not be disclosed, distributed or reproduced without the prior written permission of EPAM®.

Confidential

CONTENTS

1 INTRODUCTION

Prior to the advent of electronic data processing, companies used to manage their customers more by customer loyalty of making purchases at the same store for products, and used to track inventory using traditional bookkeeping methods. At that time, population demographics were small and buying trends for products and services were limited. When the early days of electronic data processing came about in the early 1950s, initial systems were based on punch cards. The benefit of the systems was their ability to start managing pockets of businesses in electronic formats. The downside was the proliferation of multiple stores of data on punch cards that reflected different values, and damage to the paper would mean loss of data. We quickly evolved from the punch cards to magnetic tapes that gave better data storage techniques, yet were not able to control the proliferation of different formats of data. From magnetic tapes we evolved to disks where we could store data. Along the way the applications that generated and controlled data production from the front-end perspective moved quickly from simple niche languages to declarative programming languages.

Tracking along the progress of the storage and programming languages, the applications to manage customers, employees, inventory, suppliers, finances, and sales evolved. The only issue with the data was it could not be analyzed for historical trends, as the data was updated in multiple cycles. Thus evolved the first generation of OLTP applications. Around the same time in the 1970s, Edgar. F. Codd published his paper on the relational model of systems for managing data.

Codd's paper and the release of System R, the first experimental relational database, provided the first glimpse of moving to a relational model of database systems. The subsequent emergence of multiple relational databases, such as Oracle RDB, Sybase, and SQL/DS, within a few years of the 1980s were coupled with the first editions of SQL language. OLTP systems started emerging stronger on the relational model; for the first-time companies were presented with two-tier applications where the graphical user interface (GUI) was powerful enough to model front-end needs and the underlying data was completely encapsulated from the end user.

In the late 1970s and early 1980s, the first concepts of data warehousing emerged with the need to store and analyze the data from the OLTP. The ability to gather transactions, products, services, and locations over a period started providing interesting capabilities to companies that were never there in the OLTP world, partially due to the design of the OLTP and due to the limitations with the scalability of the infrastructure.

1.1 DATA WAREHOUSES, OLTP, OLAP

Data warehouses support business decisions by collecting, consolidating, and organizing data for reporting and analysis with tools such as online analytical processing (OLAP) and data mining. Although data warehouses are built on relational database technology, the design of a data warehouse database differs substantially from the design of an online transaction processing system (OLTP) database. A relational database is designed for a specific purpose. Because the purpose of a data warehouse differs from that of an OLTP, the design characteristics of a relational database that supports a data warehouse differ from the design characteristics of an OLTP database.

Data warehouse database	Transactional database
Designed for analysis of business measures by categories and attributes	Designed for real-time business operations
Optimized for bulk loads and large, complex, unpredictable queries that access many rows per table	Optimized for a common set of transactions, usually adding, or retrieving a single row at a time per table
Loaded with consistent, valid data; requires no real time validation	Optimized for validation of incoming data during transactions; uses validation data tables
Supports few concurrent users relative to OLTP	Supports thousands of concurrent users

1.2 DESIGNING A DATA WAREHOUSE: PREREQUISITES

1.2.1 Why build a data warehouse?

Why build a data warehouse? Why is the data in an online transaction processing (OLTP) database only part of a business intelligence solution? Where does Big Data fit in a data warehousing deployment strategy?

Data warehouse relational database platforms are often designed with the following characteristics in mind:

- *Strategic and tactical analyses can discern trends in data*. Data warehouses often are used in creation of reports based on aggregate values culled from enormous amounts of data. If OLTP databases were used to create such aggregates on the fly, the database resources used would influence the ability to process transactions in a timely manner. These ad hoc queries often take advantage of computer-intensive analytic functions embedded in the database. Furthermore, if data volumes of this size were entirely pushed to in-memory databases in a middle tier, the platform cost would be prohibitive.
- *A significant portion of the data in a data warehouse is often read-only, with infrequent updates*. Database manageability features can make it possible to deploy warehouses containing petabytes of data, even where near real-time updates of some of the data is occurring.
- *The data in source systems is not "clean" or consistent across systems*. Data input to transactional systems, if not carefully controlled, is likely to contain errors and duplication. Often, a key portion of the data warehouse loading process involves elimination of these errors through data transformation. Since multiple source systems might differ in data definitions, data transformations during the ETL (extraction, transformation, and load) process can be used to modify data into a single common definition as well as improve its quality.
- *The design required for an efficient data warehouse differs from the standard normalized design for a relational database*. Queries are typically submitted against a fact table, which may contain summarized data. The schema design often used, a star schema, lets you access facts quite flexibly along key dimensions or "lookup" values. (The star schema is described in more detail later in this chapter.) For instance, a business user may want to compare the total amount of sales, which comes from a fact table, by region, store in the region, and items, all of which can be considered key dimensions. Today's data warehouses often feature a hybrid schema that is a combination of the star schema common in previous-generation data marts with third normal form schema for detailed data that is common in OLTP systems and enterprise data warehouses.
- *The data warehouse often serves as a target for meaningful data found on Big Data platforms that optimally solve semi-structured data problems*. Big Data can be described as semi-structured data containing data descriptors, data values, and other miscellaneous data bits produced by sensors, social media, and web-based data feeds. Given the amount of irrelevant data present, the processing goal on a Big Data platform is to map the data and reduce it to data of value (hence "Map Reduce" callouts in programs written using languages such as Java and Python that refine this data). This subset of Big Data is usually fed to a data warehouse where it has value across the business and might be analyzed side by side with structured data.

1.2.2 Data Warehouse Architecture Goals

A data warehouse exists to serve its users—analysts and decision makers. A data warehouse must be designed to satisfy the following requirements:

- Deliver a great user experience—user acceptance is the measure of success
- Function without interfering with OLTP systems
- Provide a central repository of consistent data
- Answer complex queries quickly
- Provide a variety of powerful analytical tools, such as OLAP and data mining

Most successful data warehouses that meet these requirements have these common characteristics:

- Are based on a dimensional model
- Contain historical data
- Include both detailed and summarized data
- Consolidate disparate data from multiple sources while retaining consistency

- Focus on a single subject, such as sales, inventory, or finance

Data warehouses are often quite large. However, size is not an architectural goal—it is a characteristic driven by the amount of data needed to serve the users.

1.2.3 Data Warehouse Users

The success of a data warehouse is measured solely by its acceptance by users. Without users, historical data might as well be archived to magnetic tape and stored in the basement. Successful data warehouse design starts with understanding the users and their needs.

Data warehouse users can be divided into four categories: Statisticians, Knowledge Workers, Information Consumers, and Executives. Each type makes up a portion of the user population as illustrated in this diagram.

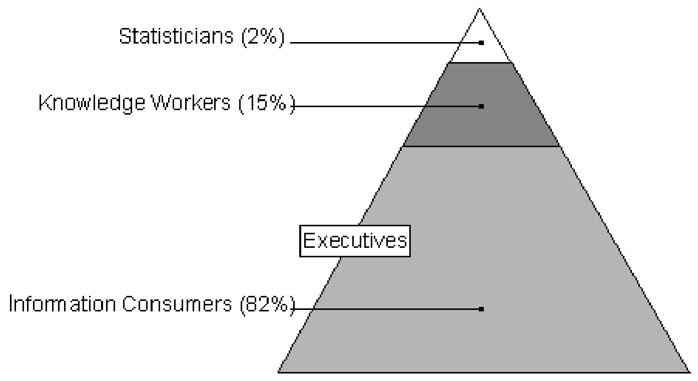

Figure 1 The User Pyramid

Statisticians: There are typically only a handful of sophisticated analysts—Statisticians and operations research types—in any organization. Though few in number, they are some of the best users of the data warehouse; those whose work can contribute to closed loop systems that deeply influence the operations and profitability of the company. It is vital that these users come to love the data warehouse. Usually that is not difficult; these people are often very self-sufficient and need only to be pointed to the database and given some simple instructions about how to get to the data and what times of the day are best for performing large queries to retrieve data to analyze using their own sophisticated tools. They can take it from there.

Knowledge Workers: A relatively small number of analysts perform the bulk of new queries and analyses against the data warehouse. These are the users who get the "Designer" or "Analyst" versions of user access tools. They will figure out how to quantify a subject area. After a few iterations, their queries and reports typically get published for the benefit of the Information Consumers. Knowledge Workers are often deeply engaged with the data warehouse design and place the greatest demands on the ongoing data warehouse operations team for training and support.

Information Consumers: Most users of the data warehouse are Information Consumers; they will probably never compose a true ad hoc query. They use static or simple interactive reports that others have developed. It is easy to forget about these users because they usually interact with the data warehouse only through the work product of others. Do not neglect these users! This group includes a large number of people, and published reports are highly visible. Set up a great communication infrastructure for distributing information widely and gather feedback from these users to improve the information sites over time.

Executives: Executives are a special case of the Information Consumers group. Few executives actually issue their own queries, but an executive's slightest musing can generate a flurry of activity among the other types of users. A wise data warehouse designer/implementer/owner will develop a very cool digital dashboard for executives, assuming it is easy and economical to do so. Usually this should follow other data warehouse work, but it never hurts to impress the bosses.

1.2.4 How Users Query the Data Warehouse

Information for users can be extracted from the data warehouse relational database or from the output of analytical services such as OLAP or data mining. Direct queries to the data warehouse relational database should be limited to those that cannot be accomplished through existing tools, which are often more efficient than direct queries and impose less load on the relational database.

Reporting tools and custom applications often access the database directly. Statisticians frequently extract data for use by special analytical tools. Analysts may write complex queries to extract and compile specific information not readily accessible through existing tools. Information consumers do not interact directly with the relational database but may receive e-mail reports or access web pages that expose data from the relational database. Executives use standard reports or ask others to create specialized reports for them.

1.3 NORMALIZATION / DENORMALIZATION

1.3.1 Normalization

Normalization is a series of steps followed to obtain a database design that allows for efficient access and storage of data. These steps reduce data redundancy and the chances of data becoming inconsistent. Normalization is an incremental process where a set of entities must first being first normal form before they can be transformed into second normal form. It follows that third normal form, can only be applied when an entity structure is in 2nd normal form, and so on. There are a number of steps in the normalization process.

1.3.2 First Normal Form

First Normal Form eliminates repeating groups by putting each into a separate table and connecting them with a one-to-many relationship.

Two rules follow this definition:

- Each table has a primary key made of one or several fields and uniquely identifying each record
- Each field is atomic, it does not contain more than one value

For instance, assuming a table WAGON to follow each wagon in every station.

Column Name	Column Type	Example Data
wagon_id	integer	(ex. 101)
description	string	(ex. 'wagon_type, empty_weight, capacity, designer, design_date')
state	string	(ex. 'under repair')
timestamp	datetime	(ex. '22/12/2008 17:37')
station	string	(ex. 'New York Grand Central')

The primary key is (wagon_id, timestamp).

This table is not in 1NF because "description" is not atomic. To move it in 1NF we have to split "description" field in its components:

Column Name	Column Type
wagon_id	integer
wagon_type	string
empty_weight	number

capacity	number
designer	string
design_date	datetime
state	string
timestamp	datetime
station	string

1.3.3 Second Normal Form

Second Normal Form eliminates functional dependencies on a partial key by putting the fields in a separate table from those that are dependent on the whole key.

In our example, "wagon_type", "empty_weight", "capacity"... only depends on "wagon_id" but not on "timestamp" field of the primary key, so this table is not in 2NF. In order to reach 2NF, we have to split the table in two in the way that each field of each table depends on all the fields of each primary key:

Column Name	Column Type
wagon_id	integer
wagon_type	string
empty_weight	number
capacity	number
designer	string
design_date	datetime

Column Name	Column Type
wagon_id	integer
timestamp	datetime
station	string
state	string

1.3.4 Third Normal Form

Third Normal Form eliminates functional dependencies on non-key fields by putting them in a separate table. At this stage, all non-key fields are dependent on the key, the whole key and nothing but the key.

In our example, in the first table it is most likely that "empty_weight", "capacity", "designer" and "design_date" depend on "wagon_type", so we have to split this table in two:

Column Name	Column Type
wagon_id	integer
wagon_type	string

Column Name	Column Type
wagon_type	string

empty_weight	number
capacity	number
designer	string
design_date	datetime

Now our example with its 3 tables is in 3NF.

1.3.5 Denormalization

By definition denormalization is simply reversing of the steps of the application of 1st to 5th normal forms applied by the normalization process. If we try to reverse above steps we will get to the first view of a table. That is denormalization.

Denormalization reintroduces duplication and, thus, decreases granularity. Being the reverse of excessive granularity, demoralizations often used to increase performance in data warehouses. Excessive normalization granularity in data warehouse databases can lead to debilitating performance problems. In addition to the denormalization of previously applied normalization, some relational databases allow for specialized objects. PostgreSQL Database allows the creation of specialized database objects largely for the purpose of speeding up query processing. One of the most effective methods of increasing query performance is by way of reducing the number of joins in queries. PostgreSQL Database allows the creation of various specialized objects just for doing this type of thing.

In general, dimensional conception designers must resist the normalization urges caused by years of operational database designs and instead denormalize the many-to-one fixed depth hierarchies into separate attributes on a flattened dimension row. Instead of third normal form, dimension tables typically are highly denormalized with flattened many-to-one relationships within a single dimension table. Because dimension tables typically are geometrically smaller than fact tables, improving storage efficiency by normalizing or snowflaking has virtually no impact on the overall database size. You should almost always trade off dimension table space for simplicity and accessibility.

There remains industry consternation about whether the data in the ETL system should be repurposed into physical normalized structures prior to loading into the presentation area’s dimensional structures for querying and reporting. The ETL system is typically dominated by the simple activities of sorting and sequential processing. In many cases, the ETL system is not based on relational technology but instead may rely on a system of flat files. After validating the data for conformance with the defined one-to-one and many-to-one business rules, it may be pointless to take the final step of building a 3NF physical database, just before transforming the data once again into denormalized structures for the BI presentation area.

2 DATA WAREHOUSE ARCHITECTURES

The following architecture properties are essential for a data warehouse system:

- Separation. Analytical and transactional processing should be kept apart as much as possible.
- Scalability. Hardware and software architectures should be easy to upgrade as the data volume, which has to be managed and processed, and the number of users’ requirements, which have to be met, progressively increase.
- Extensibility. The architecture should be able to host new applications and technologies without redesigning the whole system.
- Security. Monitoring accesses is essential because of the strategic data stored in data warehouses.
- Administerability. Data warehouse management should not be overly difficult.

Two different classifications are commonly adopted for data warehouse architectures. Separation is a structure-oriented one that depends on the number of layers used by the architecture. Scalability depends on how the different layers are employed to create enterprise-oriented or department-oriented views of data warehouses.

2.1 TYPES OF DATA WAREHOUSE ARCHITECTURE

2.1.1 Single-Layer Architecture

A single-layer architecture is not frequently used in practice. Its goal is to minimize the amount of data stored; to reach this goal, it removes data redundancies. In this case, data warehouses are virtual.

The weakness of this architecture lies in its failure to meet the requirement for separation between analytical and transactional processing. Analysis queries are submitted to operational data after the middleware interprets them. It this way, the queries affect regular transactional workloads. In addition, although this architecture can meet the requirement for integration and correctness of data, it cannot log more data than sources do. For these reasons, a virtual approach to data warehouses can be successful only if analysis needs are particularly restricted and the data volume to analyze is huge.

2.1.2 Two-Layer Architecture

The requirement for separation plays a fundamental role in defining the typical architecture for a data warehouse system, as shown in Figure 1. Although it is typically called a two-layer architecture to highlight a separation between physically available sources and data warehouses, it actually consists of five subsequent data flow stages:

1. **Source layer.** A data warehouse system uses heterogeneous sources of data. That data is originally stored to corporate relational databases or legacy databases, or it may come from information systems outside the corporate walls.
2. **Data staging.** The data stored to sources should be extracted, cleansed to remove inconsistencies and fill gaps, and integrated to merge heterogeneous sources into one common schema. The so-called Extraction, Transformation, and Loading tools (ETL) can merge heterogeneous schemata, extract, transform, cleanse, validate, filter, and load source data into a data warehouse. You must clean and process your operational data before putting it into the warehouse. You can do this programmatically, although most data warehouses use a staging area instead. A staging area simplifies building summaries and general warehouse management.
3. **Data warehouse layer.** Information is stored to one logically centralized single repository: a data warehouse. The data warehouse can be directly accessed, but it can also be used as a source for creating data marts. Meta-data repositories store information on sources, access procedures, data staging, users, data mart schemata, and so on.
4. **Data Marts.** You may want to customize your warehouse's architecture for different groups within your organization. You can do this by adding data marts, which are systems designed for a particular line of business.
5. **Analysis.** In this layer, integrated data is efficiently and flexibly accessed to issue reports, dynamically analyze information, and simulate hypothetical business scenarios. Technologically speaking, it should feature aggregate data navigators, complex query optimizers, and user-friendly GUIs.

The architectural difference between data warehouses and data marts needs to be studied closer. The component marked as a data warehouse in Figure 1 is also often called the primary data warehouse or corporate data warehouse. It acts as a centralized storage system for all the data being summed up. Data marts can be viewed as small, local data warehouses replicating (and summing up as much as possible) the part of a primary data warehouse required for a specific application domain.

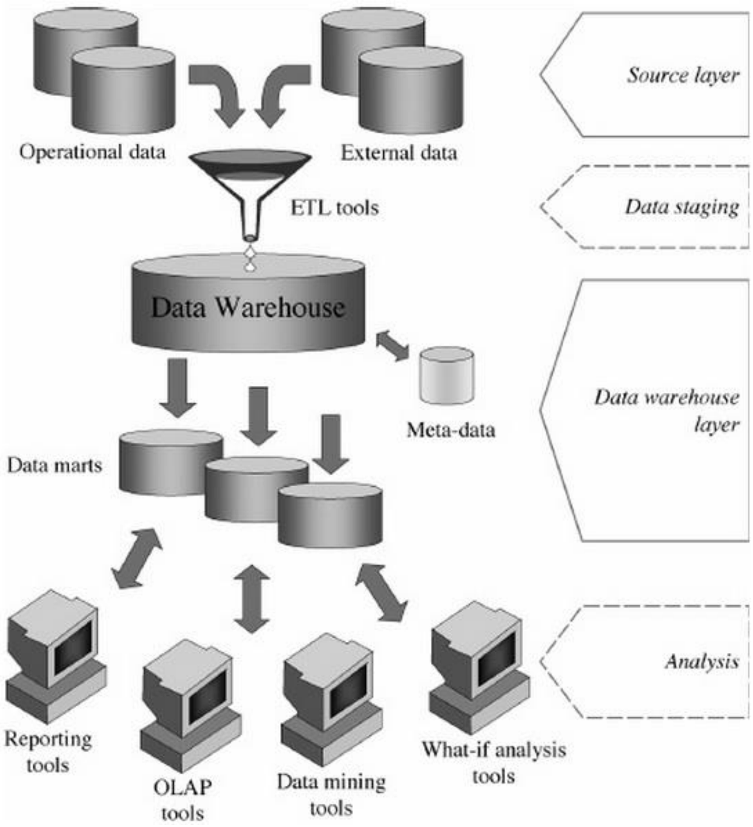

Figure 1 Two Layer DWH Architecture

Data Marts
Data marts are an important component of the front room. A data mart is a set of dimensional tables supporting a business process.

- Data marts are based on the source of data, not on a department's view of data. In other words, there is only one orders data mart in a product-oriented company. All the end user query tools and applications in various departments access this data mart to have a single, consistently labeled version of orders.
- Data marts contain all atomic detail needed to support drilling down to the lowest level. The view that data marts consist only of aggregated data is one of the most fundamental mistakes a data warehouse designer can make. Aggregated data in the absence of the lowest-level atomic data presupposes the business question and makes drilling down impossible. We will see that a data mart should consist of a continuous pyramid of identically structured dimensional tables, always beginning with the atomic data as the foundation.
- Data marts can be centrally controlled or decentralized. In other words, an enterprise data warehouse can be physically centralized on a single machine and the deployment of data marts can wait until a certain level of integration takes place in the ETL staging areas, or the data marts can be developed separately and asynchronously while at the same time participating in the enterprise's conformed dimensions and facts. We believe that the extreme of a fully centralized and fully prebuilt data warehouse is an ideal that is interesting to talk about but is not realistic. A much more realistic scenario is the incrementally developed and partially decentralized data warehouse environment. After all, organizations are constantly changing, acquiring new data sources, and needing new perspectives. So in a real environment, we must focus on incremental and adaptable strategies for building data warehouses, rather than on idealistic visions of controlling all information before a data warehouse is implemented.

2.1.3 Three-Layer Architecture

In this architecture, the third layer is the reconciled data layer or operational data store. This layer materializes operational data obtained after integrating and cleansing source data. As a result, those data are integrated, consistent, correct, current, and detailed. Figure 2 shows a data warehouse that is not populated from its sources directly, but from reconciled data.

The main advantage of the reconciled data layer is that it creates a common reference data model for a whole enterprise. At the same time, it sharply separates the problems of source data extraction and integration from those of data warehouse population. Remarkably, in some cases, the reconciled layer is also directly used to better accomplish some operational tasks, such as producing daily reports that cannot be satisfactorily prepared using the corporate applications, or generating data flows to feed external processes periodically so as to benefit from cleaning and integration. However, reconciled data leads to more redundancy of operational source data. Note that we may assume that even two-layer architectures can have a reconciled layer that is not specifically materialized, but only virtual, because it is defined as a consistent integrated view of operational source data.

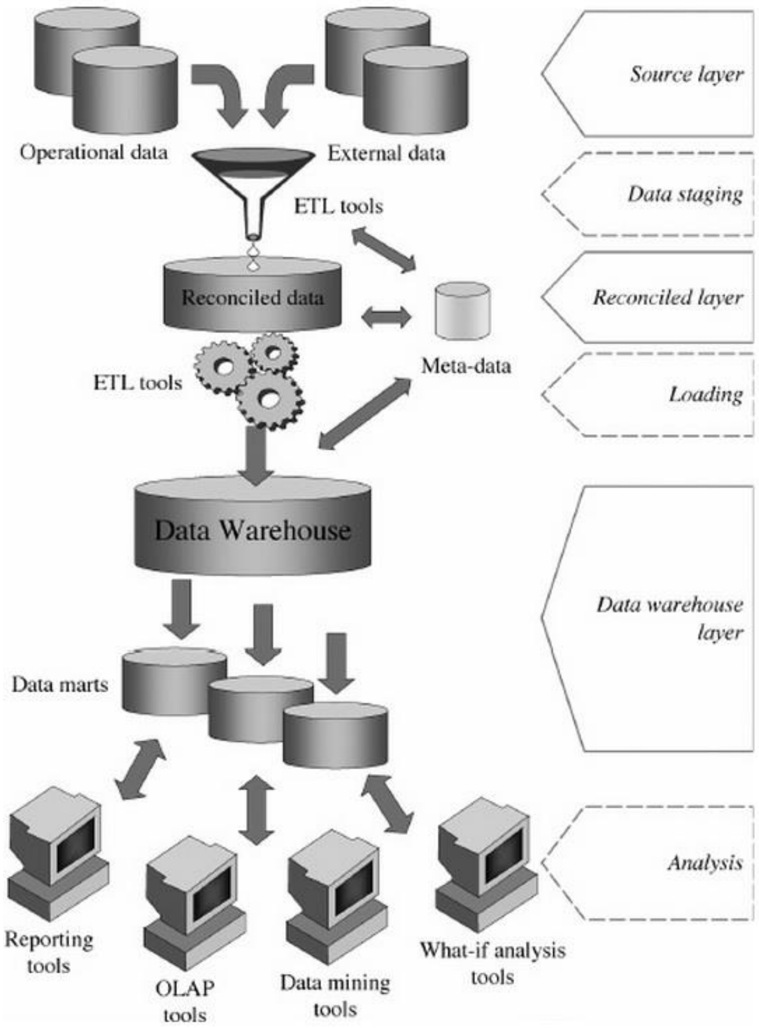

Figure 2 Three Layer DWH Architecture

2.1.4 An Additional Architecture Classification

The scientific literature often distinguishes five types of architecture for data warehouse systems, in which the same basic layers mentioned in the preceding paragraphs are combined in different ways.

In independent data marts architecture, different data marts are separately designed and built in a nonintegrated fashion. This architecture can be initially adopted in the absence of a strong sponsorship toward an enterprise-wide warehousing project, or when the organizational divisions that make up the company are loosely coupled. However, it tends to be soon replaced by other architectures that better achieve data integration and cross-reporting.

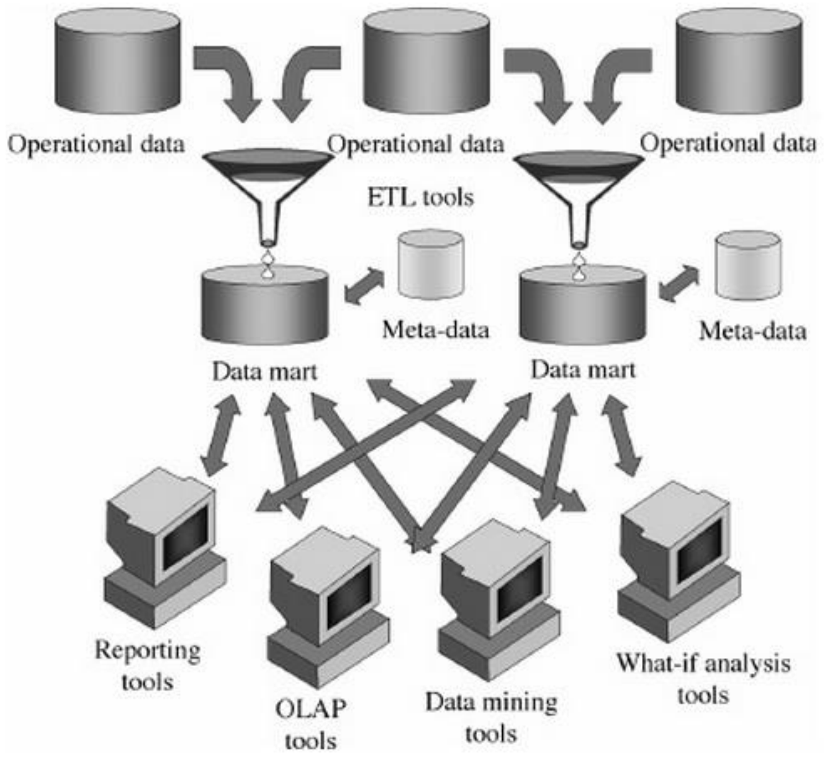

Figure 3 Independent Data Mart Architecture

The bus architecture, recommended by Ralph Kimball, is apparently similar to the preceding architecture, with one important difference. A basic set of conformed dimensions (that is, analysis dimensions that preserve the same meaning throughout all the facts they belong to), derived by a careful analysis of the main enterprise processes, is adopted and shared as a common design guideline. This ensures logical integration of data marts and an enterprise-wide view of information.

In the hub-and-spoke architecture, one of the most used in medium to large contexts, there is much attention to scalability and extensibility, and to achieving an enterprise-wide view of information. Atomic, normalized data is stored in a reconciled layer that feeds a set of data marts containing summarized data in multidimensional form (Figure 4). Users mainly access the data marts, but they may occasionally query the reconciled layer.

The centralized architecture, recommended by Bill Inmon, can be seen as a particular implementation of the hub-and-spoke architecture, where the reconciled layer and the data marts are collapsed into a single physical repository.

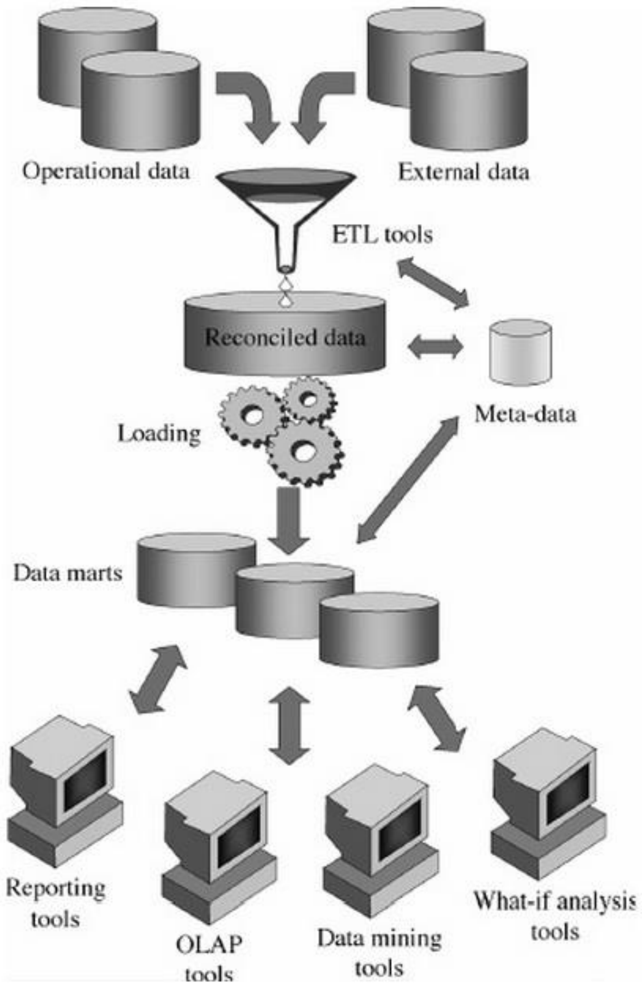

Figure 4 Hub-and-spoke Architecture

The federated architecture is sometimes adopted in dynamic contexts where preexisting data warehouses/data marts are to be noninvasively integrated to provide a single, cross-organization decision support environment (for instance, in the case of mergers and acquisitions). Each data warehouse/data mart is either virtually or physically integrated with the others, leaning on a variety of advanced techniques such as distributed querying, ontologies, and meta-data interoperability.

The following list includes the factors that are particularly influential when it comes to choosing one of these architectures:

- The amount of interdependent information exchanged between organizational units in an enterprise and the organizational role played by the data warehouse project sponsor may lead to the implementation of enterprise-wide architectures, such as bus architectures, or department-specific architectures, such as independent data marts.
- An urgent need for a data warehouse project, restrictions on economic and human resources, as well as poor IT staff skills may suggest that a type of “quick” architecture, such as independent data marts, should be implemented.
- The minor role played by a data warehouse project in enterprise strategies can make you prefer an architecture type based on independent data marts over a hub-and-spoke architecture type.
- The frequent need for integrating preexisting data warehouses, possibly deployed on heterogeneous platforms, and the pressing demand for uniformly accessing their data can require a federated architecture type.

2.2 PHYSICAL MODEL

The starting point for the physical model is the logical model. The physical model should mirror the logical model as much as possible, although some changes in the structure of the tables and / or columns may be necessary. In addition the physical model will include staging or maintenance tables that are usually not included in the logical model. Although your environment may not have such clearly defined layers you should have some aspects of each layer in your database to ensure it will continue to scale as it increases in size and complexity.

2.2.1 Staging layer

The staging layer enables the speedy extraction, transformation and loading (ETL) of data from your operational systems into the data warehouse without distributing any of the business users. The tables in the staging layer are normally segregated from the "live" data warehouse. The most basic approach for the staging layer is to have it be an identical schema to the one that exists in the source operational system(s) but with some structural changes to the tables, such as range partitioning. It is also possible that in some implementations this layer is not necessary, as all data transformation processing will be done "on the fly" as data is extracted from the source system before it is inserted directly into the Foundation Layer. Either way you will still have to load data into the warehouse.

2.2.1.1 Efficient Data Loading

Whether you are loading into a staging layer or directly into the foundation layer the goal is to get the data into the warehouse in the most expedient manner. In order to achieve good performance during the load you need to begin by focusing on where the data to-being-loaded resides and how you load it into the database. For example, you should not use a serial database link or a single JDBC connection to move large volumes of data. Flat files are the most common and preferred mechanism of large volumes of data.

2.2.1.2 Staging Area

The area where flat files are stored prior to being loaded into the staging layer of a data warehouse system is commonly known as staging area. The overall speed of your load will be determined by (A) how quickly the raw data can be read from staging area and (B) how fast it can be processed and inserted into the database. It is highly recommended that you stage the raw data across as many physical disks as possible to ensure the reading it is not a bottleneck during the load.

2.2.1.3 Performance tips

One might need to insert a large amount of data when first populating a database. This section contains some suggestions on how to make this process as efficient as possible.

When using multiple `INSERT`s, turn off autocommit and just do one commit at the end. (In plain SQL, this means issuing `BEGIN` at the start and `COMMIT` at the end. Some client libraries might do this behind your back, in which case you need to make sure the library does it when you want it done.) If you allow each insertion to be committed separately, PostgreSQL is doing a lot of work for each row that is added. An additional benefit of doing all insertions in one transaction is that if the insertion of one row were to fail then the insertion of all rows inserted up to that point would be rolled back, so you won't be stuck with partially loaded data.

Use `COPY` to load all the rows in one command, instead of using a series of `INSERT` commands. The `COPY` command is optimized for loading large numbers of rows; it is less flexible than `INSERT`, but incurs significantly less overhead for large data loads. Since `COPY` is a single command, there is no need to disable autocommit if you use this method to populate a table.

If you cannot use `COPY`, it might help to use `PREPARE` to create a prepared `INSERT` statement, and then use `EXECUTE` as many times as required. This avoids some of the overhead of repeatedly parsing and planning `INSERT`. Different interfaces provide this facility in different ways; look for "prepared statements" in the interface documentation.

Note that loading a large number of rows using `COPY` is almost always faster than using `INSERT`, even if `PREPARE` is used and multiple insertions are batched into a single transaction.

If you are loading a freshly created table, the fastest method is to create the table, bulk load the table's data using `COPY`, then create any indexes needed for the table. Creating an index on pre-existing data is quicker than updating it incrementally as each row is loaded.

If you are adding large amounts of data to an existing table, it might be a win to drop the indexes, load the table, and then recreate the indexes. Of course, the database performance for other users might suffer during the time the indexes are missing. One should also think twice before dropping a unique index, since the error checking afforded by the unique constraint will be lost while the index is missing.

Just as with indexes, a foreign key constraint can be checked "in bulk" more efficiently than row-by-row. So it might be useful to drop foreign key constraints, load data, and re-create the constraints. Again, there is a trade-off between data load speed and loss of error checking while the constraint is missing.

What's more, when you load data into a table with existing foreign key constraints, each new row requires an entry in the server's list of pending trigger events (since it is the firing of a trigger that checks the row's foreign key constraint). Loading many millions of rows can cause the trigger event queue to overflow available memory, leading to intolerable swapping or even outright failure of the command. Therefore, it may be *necessary*, not just desirable, to drop and re-apply foreign keys when loading large amounts of data. If temporarily removing the constraint isn't acceptable, the only other recourse may be to split up the load operation into smaller transactions.

2.2.1.4 Foreign data

PostgreSQL implements portions of the SQL/MED specification, allowing you to access data that resides outside PostgreSQL using regular SQL queries. Such data is referred to as *foreign data*. (Note that this usage is not to be confused with foreign keys, which are a type of constraint within the database.)

Foreign data is accessed with help from a *foreign data wrapper*. A foreign data wrapper is a library that can communicate with an external data source, hiding the details of connecting to the data source and obtaining data from it. There are some foreign data wrappers available as `contrib` modules;

Other kinds of foreign data wrappers might be found as third party products. If none of the existing foreign data wrappers suit your needs, you can write your own;

To access foreign data, you need to create a *foreign server* object, which defines how to connect to a particular external data source according to the set of options used by its supporting foreign data wrapper. Then you need to create one or more *foreign tables*, which define the structure of the remote data. A foreign table can be used in queries just like a normal table, but a foreign table has no storage in the PostgreSQL server. Whenever it is used, PostgreSQL asks the foreign data wrapper to fetch data from the external source, or transmit data to the external source in the case of update commands.

Accessing remote data may require authenticating to the external data source. This information can be provided by a *user mapping*, which can provide additional data such as user names and passwords based on the current PostgreSQL role.

2.2.1.5 Data Compression

Another key decision that you need to make during the load phase is whether or not to compress your data. Using table compression obviously reduces disk and memory usage, often resulting in better scale-up performance for read-only operations. Table compression can also speed up query execution by minimizing the number of round trips required to retrieve data from the disks. Compressing data however imposes a performance penalty on the load speed of the data. The overall performance gain typically outweighs the cost of compression.

If you decide to use compression, consider sorting your data before loading it to achieve the best possible compression rate. The easiest way to sort incoming data is to load it using an ORDER BY clause on either your CTAS or IAS statement. You should ORDER BY a NOT NULL column (ideally non numeric) that has a large number of distinct values (1,000 to 10,000).

2.2.2 Foundation layer - Third Normal Form

From staging, the data will transition into the foundation or integration layer via another set of ETL processes. Data begins to take shape and it is not uncommon to have some end-user application access data from this layer especially if they are time sensitive, as data will become available here before it is transformed into the dimension / performance layer. Traditionally this layer is implemented in the Third Normal Form (3NF).

There are several approaches to optimize 3NF schema. The larger tables should be partitioned using composite partitioning (range-hash or list-hash). There are three reasons for this:

1. Easier manageability of terabytes of data
2. Faster accessibility to the necessary data
3. Efficient and performant table joins

Finally Parallel Execution enables a database task to be parallelized or divided into smaller units of work, thus allowing multiple processes to work concurrently. By using parallelism, a terabyte of data can be scanned and processed in minutes or less, not hours or days.

2.2.3 Access layer - Star Schema

The access layer represents data, which is in a form that most users and applications can understand. It is in this layer you are most likely to see a star schema.

Tuning a star query is very straight forward. The two most important criteria are:

- Create a bitmap index on each of the foreign key columns in the fact table or tables
- Set the initialization parameter STAR_TRANSFORMATION_ENABLED to TRUE. This will enable the optimizer feature for star queries which is off by default for backward compatibility.

2.2.4 Meta-data

The term meta-data can be applied to the data used to define other data. In the scope of data warehousing, meta-data plays an essential role because it specifies source, values, usage, and features of data warehouse data and defines how data can be changed and processed at every architecture layer. Figures 1 and 2 show that the meta-data repository is closely connected to the data warehouse. Actually, according to Kimball, metadata is everything, except for the data itself. Applications use it intensively to carry out data-staging and analysis tasks.

You can classify meta-data into two partially overlapping categories. This classification is based on the ways system administrators and end users exploit meta-data. System administrators are interested in internal meta-data (back room metadata) because it defines data sources, transformation processes, population policies, logical and physical schemata, constraints, and user profiles. External meta-(front room metadata) data is relevant to end users. For example, it is about definitions, quality standards, units of measure, relevant aggregations.

3 ACCESSING DATA WAREHOUSES

Analysis is the last level common to all data warehouse architecture types. After cleansing, integrating, and transforming data, you should determine how to get the best out of it in terms of information. The following sections show the best approaches for end users to query data warehouses: reports, OLAP, and dashboards. End users often use the information stored to a data warehouse as a starting point for additional business intelligence applications, such as what-if analyses and data mining.

3.1 REPORTS

This approach is oriented to those users who need to have regular access to the information in an almost static way. For example, suppose a local health authority must send to its state offices monthly reports summing up information on patient admission costs. The layout of those reports has been predetermined and may vary only if changes are applied to current laws and regulations. Designers issue the queries to create reports with the desired layout and “freeze” all those in an application. In this way, end users can query current data whenever they need to.

A report is defined by a query and a layout. A query generally implies a restriction and an aggregation of multidimensional data. For example, you can look for the monthly receipts during the last quarter for every product category. A layout can look like a table or a chart (diagrams, histograms, pies, and so on).

A reporting tool should be evaluated not only on the basis of comprehensive report layouts, but also on the basis of flexible report delivery systems. A report can be explicitly run by users or automatically and regularly sent to registered end users. For example, it can be sent via e-mail.

Keep in mind that reports existed long before data warehouse systems came to be. Reports have always been the main tool used by managers for evaluating and planning tasks since the invention of databases. However, adding data warehouses to the mix is beneficial to reports for two main reasons: First, they take advantage of reliable and correct results because the data summed up in reports is consistent and integrated. In addition, data warehouses expedite the reporting process because the architectural separation between transaction processing and analyses significantly improves performance.

3.2 OLAP

OLAP might be the main way to exploit information in a data warehouse. Surely it is the most popular one, and it gives end users, whose analysis needs are not easy to define beforehand, the opportunity to analyze and explore data interactively on the basis of the multidimensional model. While users of reporting tools essentially play a passive role, OLAP users are able to start a complex analysis session actively, where each step is the result of the outcome of preceding steps. Real-time properties of OLAP sessions, required in-depth knowledge of data, complex queries that can be issued, and design for users not familiar with IT make the tools in use play a crucial role. The GUI of these tools must be flexible, easy-to-use, and effective.

An OLAP session consists of a navigation path that corresponds to an analysis process for facts according to different viewpoints and at different detail levels. This path is turned into a sequence of queries, which are often not issued directly, but differentially expressed with reference to the previous query. The results of queries are multidimensional. Because we humans have a difficult time deciphering diagrams of more than three dimensions, OLAP tools typically use tables to display data, with multiple headers, colors, and other features to highlight data dimensions.

3.3 DASHBOARDS

Dashboards are another method used for displaying information stored to a data warehouse. The term dashboard refers to a GUI that displays a limited amount of relevant data in a brief and easy-to-read format. Dashboards can provide a real-time overview of the trends for a specific phenomenon or for many phenomena that are strictly connected with each other. The term is a visual metaphor: the group of indicators in the GUI are displayed like a car dashboard. Dashboards are often used by senior managers who need a quick way to view information. However, to conduct and display very complex analyses of phenomena, dashboards must be matched with analysis tools. Today, most software vendors offer dashboards for report creation and display.

Keep in mind, however, that dashboards are nothing but performance indicators behind GUIs. Their effectiveness is due to a careful selection of the relevant measures, while using data warehouse information quality standards. For this reason, dashboards should be viewed as a sophisticated effective add-on to data warehouse systems, but not as the primary goal of data warehouse systems. In fact, the primary goal of data warehouse systems should always be to properly define a process to transform data into information.

4 OLAP, MOLAP, AND HOLAP

These three acronyms conceal three major approaches to implementing data warehouses, and they are related to the logical model used to represent data:

- ROLAP stands for Relational OLAP, an implementation based on relational DBMSs.
- MOLAP stands for Multidimensional OLAP, an implementation based on multidimensional DBMSs.
- HOLAP stands for Hybrid OLAP, an implementation using both relational and multidimensional techniques.

The idea of adopting the relational technology to store data to a data warehouse has a solid foundation if you consider the huge amount of literature written about the relational model, the broadly available corporate experience with relational database usage and management, and the top performance and flexibility standards of relational DBMSs (RDBMSs). The expressive power of the relational model, however, does not include the concepts of dimension, measure, and hierarchy, so you must create specific types of schemata so that you can represent the multidimensional model in terms of basic relational elements such as attributes, relations, and integrity constraints. This task is mainly performed by the well-known star schema.

The main problem with ROLAP implementations results from the performance hit caused by costly join operations between large tables. To reduce the number of joins, one of the key concepts of ROLAP is denormalization—a conscious breach in the third normal form oriented to performance maximization. To minimize execution costs, the other key word is redundancy, which is the result of the materialization of some derived tables (views) that store aggregate data used for typical OLAP queries.

Different from a ROLAP system, a MOLAP system is based on an ad hoc logical model that can be used to represent multidimensional data and operations directly. The underlying multidimensional database physically stores data as arrays and the access to it is positional. Grid-files (Nievergelt et al., 1984; Whang and Krishnamurthy, 1991), R*-trees (Beckmann et al., 1990) and UB-trees (Markl et al., 2001) are among the techniques used for this purpose.

The greatest advantage of MOLAP systems in comparison with ROLAP is that multidimensional operations can be performed in an easy, natural way with MOLAP without any need for complex join operations. For this reason, MOLAP system performance is excellent.

The intermediate architecture type, HOLAP, aims at mixing the advantages of both basic solutions. It takes advantage of the standardization level and the ability to manage large amounts of data from ROLAP implementations, and the query speed typical of MOLAP systems. HOLAP implies that the largest amount of data should be stored in an RDBMS to avoid the problems caused by sparsity, and that a multidimensional system stores only the information users most frequently need to access. If that information is not enough to solve queries, the system will transparently access the part of the data managed by the relational system.

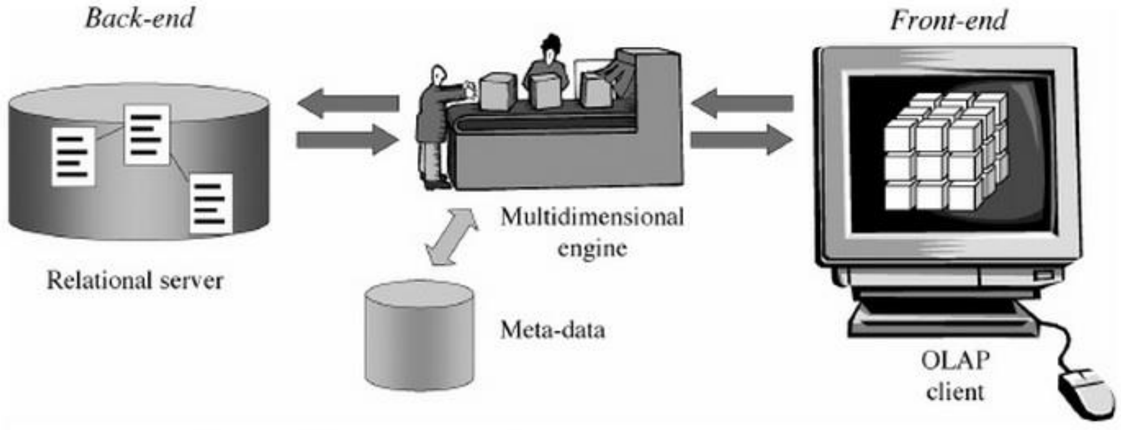

Figure 5 ROLAP Architecture

5 ADDITIONAL ISSUES

The issues that follow can play a fundamental role in tuning up a data warehouse system. These points involve very wide-ranging problems and are mentioned here to give you the most comprehensive picture possible.

5.1 QUALITY

In general, we can say that the quality of a process stands for the way a process meets users' goals. In data warehouse systems, quality is not only useful for the level of data, but above all for the whole integrated system, because of the goals and usage of data warehouses. A strict quality standard must be ensured from the first phases of the data warehouse project.

Defining, measuring, and maximizing the quality of a data warehouse system can be very complex problems. For this reason, we mention only a few properties characterizing data quality here:

- Accuracy. Stored values should be compliant with real-world ones.
- Freshness. Data should not be old.
- Completeness. There should be no lack of information.
- Consistency. Data representation should be uniform.
- Availability. Users should have easy access to data.
- Traceability. Data can easily be traced data back to its sources.
- Clearness. Data can be easily understood.

Technically, checking for data quality requires appropriate sets of metrics. In the following sections, we provide an example of the metrics for a few of the quality properties mentioned:

- Accuracy and completeness. Refers to the percentage of tuples not loaded by an ETL process and categorized on the basis of the types of problem arising. This property shows the percentage of missing, invalid, and nonstandard values of every attribute.
- Freshness. Defines the time elapsed between the date when an event takes place and the date when users can access it.
- Consistency. Defines the percentage of tuples that meet business rules that can be set for measures of an individual cube or many cubes and the percentage of tuples meeting structural constraints imposed by the data model (for example, uniqueness of primary keys, referential integrity, and cardinality constraint compliance).

Note that corporate organization plays a fundamental role in reaching data quality goals. This role can be effectively played only by creating an appropriate and accurate certification system that defines a limited group of users in charge of data. For this reason, designers must raise senior managers' awareness of this topic. Designers must also motivate management to create an accurate certification procedure specifically differentiated for every enterprise area. A board of corporate managers promoting data quality may trigger a virtuous cycle that is more powerful and less costly than any data cleansing solution. For example, you can achieve awesome results if you connect a corporate department budget to a specific data quality threshold to be reached.

An additional topic connected to the quality of a data warehouse project is related to documentation. Today most documentation is still nonstandardized. It is often issued at the end of the entire data warehouse project. Designers and implementers consider documentation a waste of time, and data warehouse project customers consider it an extra cost item. Software engineering teaches that a standard system for documents should be issued, managed, and validated in compliance with project deadlines. This system can ensure that different data warehouse project phases are correctly carried out and that all analysis and implementation points are properly examined and understood. In the medium and long term, correct documents increase the chances of reusing data warehouse projects and ensure project know-how maintenance.

5.2 SECURITY

Information security is generally a fundamental requirement for a system, and it should be carefully considered in software engineering at every project development stage from requirement analysis through implementation to maintenance. Security is particularly relevant to data warehouse projects, because data warehouses are used to manage information crucial for strategic decision-making processes. Furthermore, multidimensional properties and aggregation cause additional security problems similar to those that generally arise in statistic databases, because they implicitly offer the opportunity to infer information from data. Finally, the huge amount of information exchange that takes place in data warehouses in the data-staging phase causes specific problems related to network security.

Appropriate management and auditing control systems are important for data warehouses. Management control systems can be implemented in front-end tools or can exploit operating system services. As far as auditing is concerned, the techniques provided by DBMS servers are not generally appropriate for this scope. For this reason, you must take advantage of the systems implemented by OLAP engines. From the viewpoint of users' profile–based data access, basic requirements are related to hiding whole cubes, specific cube slices, and specific cube measures. Sometimes you also have to hide cube data beyond a given detail level.

5.3 EVOLUTION

Many mature data warehouse implementations are currently running in midsize and large companies. The unstoppable evolution of application domains highlights dynamic features of data warehouses connected to the way information changes at two different levels as time goes by:

- Data level. Even if measured data is naturally logged in data warehouses thanks to temporal dimensions marking events, the multidimensional model implicitly assumes that hierarchies are completely static. It is clear that this assumption is not very realistic. For example, a company can add new product categories to its catalog and remove others, or it can change the category to which an existing product belongs in order to meet new marketing strategies.
- Schema level. A data warehouse schema can vary to meet new business domain standards, new users' requirements, or changes in data sources. New attributes and measures can become necessary. For example, you can add a subcategory to a product hierarchy to make analyses richer in detail. You should also consider that the set of fact dimensions can vary as time goes by.

Temporal problems are even more challenging in data warehouses than in operational databases, because queries often cover longer periods. For this reason, data warehouse queries frequently deal with different data and/or schema versions. Moreover, this point is particularly critical for data warehouses that run for a long time, because every evolution not completely controlled causes a growing gap between the real world and its database representation, eventually making the data warehouses obsolete and useless.

6 THE INTERIM RESULTS

To compare effectiveness and success of different architectures implementations a web-based survey—targeted at individuals involved in an organization's data warehouse implementation used to collect data. The survey included questions about the respondent, the respondent's company, the company's data warehouse, and the success of the data warehouse architecture. Four hundred fifty-four respondents provided usable information.

Surveyed companies ranged from small (less than $10 million in revenue) to large (in excess of $10 billion). Most of the companies are located in the United States (60%) and represent a variety of industries, with the financial services industry (15%) providing the most responses.

The predominant architecture was the hub-and-spoke (39%), followed by the bus architecture (26%), centralized (17%), independent data marts (12%), and federated (4%). The most common platform for hosting the data warehouses was Oracle (41%), followed by Microsoft (19%) and IBM (18%). The average (mean) gross revenue varied from $3.7 billion for independent data marts to $6 billion for the federated architecture.

Four measures were used to assess the success of the architectures: (1) information quality, (2) system quality, (3) individual impacts, and (4) organizational impacts. The questions used a seven-point scale, with the higher score indicating a more successful architecture. Figure 7 shows the average scores for the measures across the architectures

Independent data marts scored the lowest on all measures. This finding confirms the conventional wisdom that independent data marts are a poor architectural solution.

Next lowest on all measures was the federated architecture. Firms sometimes have disparate decision-support platforms resulting from mergers and acquisitions, and they may choose a federated approach, at least in the short run. The findings suggest that the federated architecture is not an optimal long-term solution.

What is interesting, however, is the similarity of the averages for the bus, hub-and-spoke, and centralized architectures. The differences are sufficiently small that no claims can be made for a particular architecture's superiority over the others, at least based on a simple comparison of these success measures.

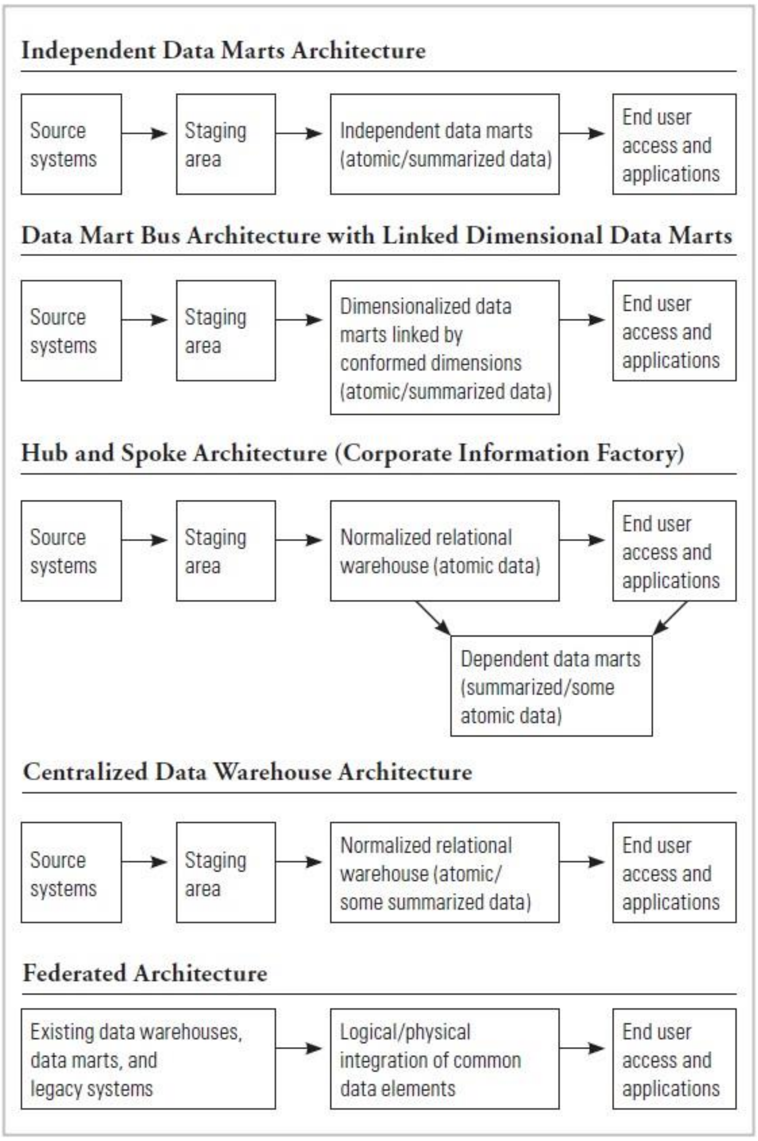
Independent Data Marts Architecture
Source systems
Staging area
Independent data marts (atomic/summarized data)
End user access and applications
Data Mart Bus Architecture with Linked Dimensional Data Marts
Source systems
Staging area
Dimensionalized data marts linked by conformed dimensions (atomic/summarized data)
End user access and applications
Hub and Spoke Architecture (Corporate Information Factory)
Source systems
Staging area
Normalized relational warehouse (atomic data)
End user access and applications
Dependent data marts (summarized/some atomic data)
Centralized Data Warehouse Architecture
Source systems
Staging area
Normalized relational warehouse (atomic/ some summarized data)
End user access and applications
Federated Architecture
Existing data warehouses, data marts, and legacy systems
Logical/physical integration of common data elements
End user access and applications

Figure 6 The five data warehouse architectures studied.

	Independent Data Marts	Bus Architecture	Hub and Spoke	Centralized (No Dependent Data Marts)	Federated
Information Quality	4.42	5.16	5.35	5.23	4.73
System Quality	4.59	5.60	5.56	5.41	4.69
Individual Impacts	5.08	5.80	5.62	5.64	5.15
Organizational Impacts	4.66	5.34	5.24	5.30	4.77

Figure 7 The success of the five architectures.

The similar success of the bus, hub-and-spoke, and centralized architectures is perhaps not all that surprising. In some ways, the architectures have evolved over time and become more similar. For example, the hub-and-spoke architecture often includes dimensional data marts, which is at the heart of the bus architecture. Even the development methodologies (e.g., top down for the hub-and-spoke and centralized architectures, and life cycle or bottom up for the bus architecture) have evolved and become more similar. Each stresses the need to start small and deliver short-term “wins” but have a long-term plan. This evolution is appropriate and good for the industry, but it is also a likely reason that the scores on the success metrics are similar.

7 SOURCE BOOKS AND ARTICLES

1. Browning, D. & Mundy, J. Microsoft Corporation (2001). *Data Warehouse Design Considerations*. http://technet.microsoft.com/en-us/library/aa902672(v=sql.80).aspx (Accessed 2014-02-10).
2. Kimball R., & Ross M. *The Data Warehouse Toolkit: The Definitive Guide to Dimensional Modeling, Third Edition*. Indianapolis: John Wiley & Sons, 2013.
3. *Normalization*. http://www.orafaq.com/wiki/Normalization. (Accessed 2014-02-10).
4. Golfarelli M., & Rizzi S. Data Warehouse Design: Modern Principles and Methodologies. New York City: McGraw-Hill Osborne Media, 2009.
5. Watson H.J., & Ariyachandra T. Data Warehouse Architectures: Factors in the Selection Decision and the Success of the Architectures, 2005.