Building the Data Warehouse, Third Edition

Table of Contents

Chapter 1: Evolution of Decision Support Systems

Overview

We are told that the hieroglyphics in Egypt are primarily the work of an accountant declaring how much grain is owed the Pharaoh. Some of the streets in Rome were laid out by civil engineers more than 2,000 years ago. Examination of bones found in archeological excavations shows that medicine—in, at least, a rudimentary form—was practiced as long as 10,000 years ago. Other professions have roots that can be traced back to antiquity. From this perspective, the profession and practice of information systems and processing is certainly immature, because it has existed only since the early 1960s.

Information processing shows this immaturity in many ways, such as its tendency to dwell on detail. There is the notion that if we get the details right, the end result will somehow take care of itself and we will achieve success. It's like saying that if we know how to lay concrete, how to drill, and how to install nuts and bolts, we don't have to worry about the shape or the use of the bridge we are building. Such an attitude would drive a more professionally mature civil engineer crazy. Getting all the details right does not necessarily bring more success.

The data warehouse requires an architecture that begins by looking at the whole and then works down to the particulars. Certainly, details are important throughout the data warehouse. But details are important only when viewed in a broader context.

The story of the data warehouse begins with the evolution of information and decision support systems. This broad view should help put data warehousing into clearer perspective.

The Evolution

The origins of DSS processing hark back to the very early days of computers and information systems. It is interesting that decision support system (DSS) processing developed out of a long and complex evolution of information technology. Its evolution continues today.

Figure 1.1 shows the evolution of information processing from the early 1960s up to 1980. In the early 1960s, the world of computation consisted of creating individual applications that were run using master files. The applications featured reports and programs, usually built in COBOL. Punched cards were common. The master files were housed on magnetic tape, which were good for storing a large volume of data cheaply, but the drawback was that they had to be accessed sequentially. In a given pass of a magnetic tape file, where 100 percent of the records have to be accessed, typically only 5 percent or fewer of the records are actually needed. In addition, accessing an entire tape file may take as long as 20 to 30 minutes, depending on the data on the file and the processing that is done.

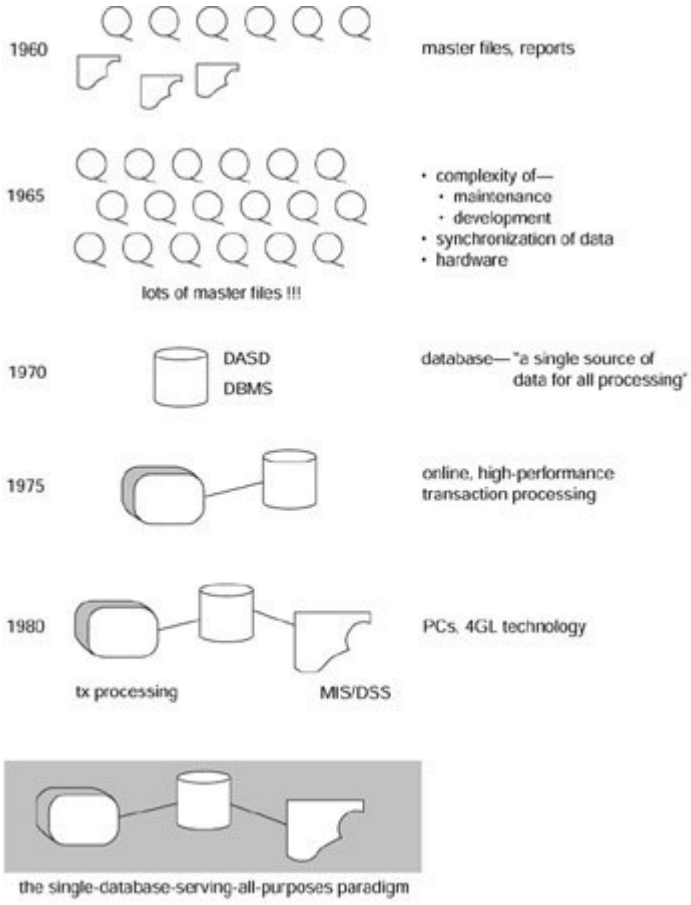

Figure 1.1: The early evolutionary stages of the architected environment.

Around the mid-1960s, the growth of master files and magnetic tape exploded. And with that growth came huge amounts of redundant data. The proliferation of master files and redundant data presented some very insidious problems:

- The need to synchronize data upon update
- The complexity of maintaining programs
- The complexity of developing new programs
- The need for extensive amounts of hardware to support all the master files

In short order, the problems of master files—problems inherent to the medium itself—became stifling.

It is interesting to speculate what the world of information processing would look like if the only medium for storing data had been the magnetic tape. If there had never been anything to store bulk data on other than magnetic tape files, the world would have never had large, fast reservations systems, ATM systems, and the like. Indeed, the ability to store and manage data on new kinds of media opened up the way for a more powerful type of processing that brought the technician and the businessperson together as never before.

The Advent of DASD

By 1970, the day of a new technology for the storage and access of data had dawned. The 1970s saw the advent of disk storage, or direct access storage device (DASD). Disk storage was fundamentally different from magnetic tape storage in that data could be accessed

directly on DASD. There was no need to go through records 1, 2, 3, ... *n* to get to record *n* 1 1. Once the address of record *n* 1 1 was known, it was a simple matter to go to record *n* 1 1 directly. Furthermore, the time required to go to record *n* 1 1 was significantly less than the time required to scan a tape. In fact, the time to locate a record on DASD could be measured in milliseconds.

With DASD came a new type of system software known as a database management system (DBMS). The purpose of the DBMS was to make it easy for the programmer to store and access data on DASD. In addition, the DBMS took care of such tasks as storing data on DASD, indexing data, and so forth. With DASD and DBMS came a technological solution to the problems of master files. And with the DBMS came the notion of a "database." In looking at the mess that was created by master files and the masses of redundant data aggregated on them, it is no wonder that in the 1970s a database was defined as a single source of data for all processing.

By the mid-1970s, online transaction processing (OLTP) made even faster access to data possible, opening whole new vistas for business and processing. The computer could now be used for tasks not previously possible, including driving reservations systems, bank teller systems, manufacturing control systems, and the like. Had the world remained in a magnetic-tape-file state, most of the systems that we take for granted today would not have been possible.

PC/4GL Technology

By the 1980s, more new technologies, such as PCs and fourth-generation languages (4GLs), began to surface. The end user began to assume a role previously unfathomed—directly controlling data and systems—a role previously reserved for the data processor. With PCs and 4GL technology came the notion that more could be done with data than simply processing online transactions. MIS (management information systems), as it was called in the early days, could also be implemented. Today known as DSS, MIS was processing used to drive management decisions. Previously, data and technology were used exclusively to drive detailed operational decisions. No single database could serve both operational transaction processing and analytical processing at the same time. Figure 1.1 shows the single-database paradigm.

Enter the Extract Program

Shortly after the advent of massive OLTP systems, an innocuous program for "extract" processing began to appear (see Figure 1.2).

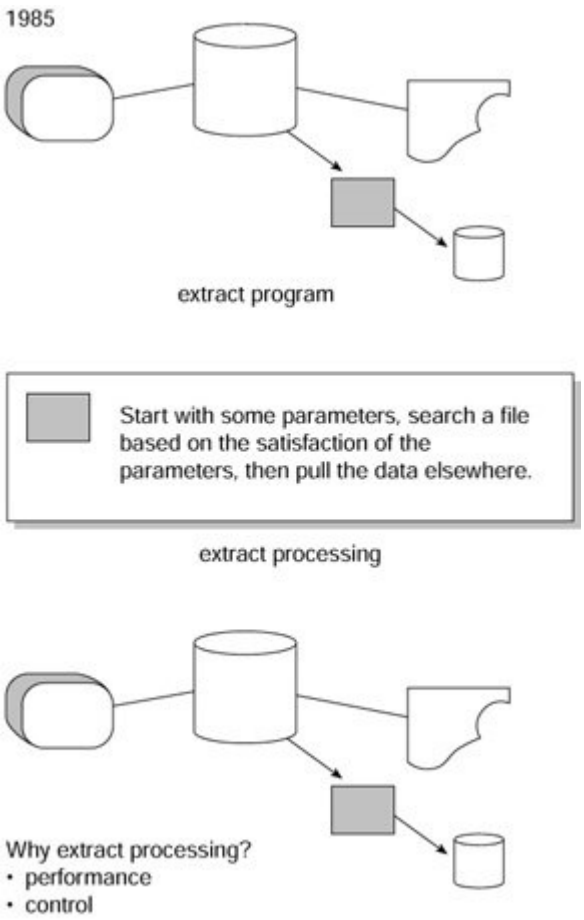

Figure 1.2: The nature of extract processing.

The extract program is the simplest of all programs. It rummages through a file or database, uses some criteria for selecting data, and, on finding qualified data, transports the data to another file or database.

The extract program became very popular, for at least two reasons:

- Because extract processing can move data out of the way of high-performance online processing, there is no conflict in terms of performance when the data needs to be analyzed en masse.
- When data is moved out of the operational, transaction-processing domain with an extract program, a shift in control of the data occurs. The end user then owns the data once he or she takes control of it. For these (and probably a host of other) reasons, extract processing was soon found everywhere.

The Spider Web

As illustrated in Figure 1.3, a “spider web” of extract processing began to form. First, there were extracts; then there were extracts of extracts; then extracts of extracts of extracts; and so forth. It was not unusual for a large company to perform as many as 45,000 extracts per day.

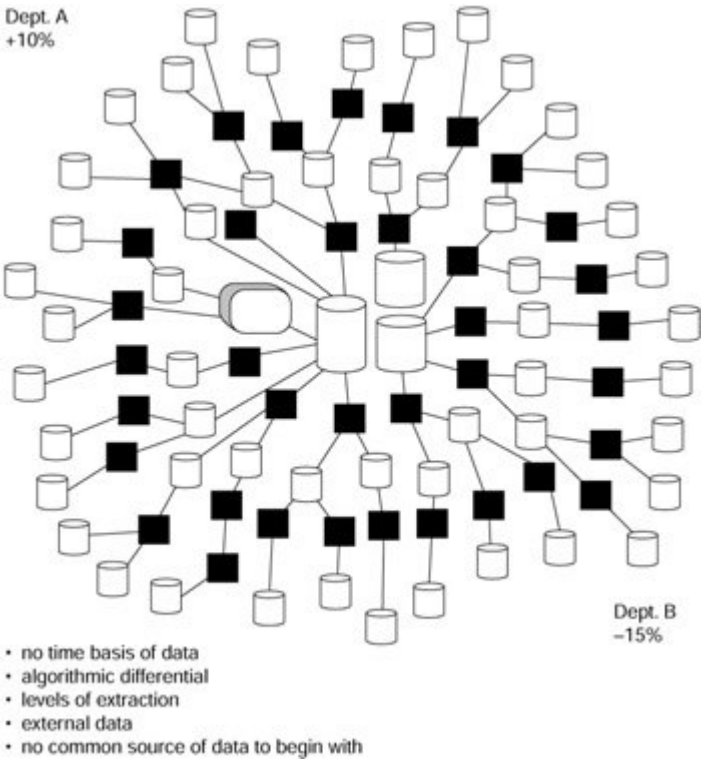

Figure 1.3: Lack of data credibility in the naturally evolving architecture.

This pattern of out-of-control extract processing across the organization became so commonplace that it was given its own name—the "naturally evolving architecture"—which occurs when an organization handles the whole process of hardware and software architecture with a laissez-faire attitude. The larger and more mature the organization, the worse the problems of the naturally evolving architecture become.

Problems with the Naturally Evolving Architecture

The naturally evolving architecture presents many challenges, such as:

- Data credibility
- Productivity
- Inability to transform data into information

Lack of Data Credibility

The lack of data credibility is illustrated in Figure 1.3. Say two departments are delivering a report to management—one department claims that activity is down 15 percent, the other says that activity is up 10 percent. Not only are the two departments not in sync with each other, they are off by very large margins. In addition, trying to reconcile the departments is difficult. Unless very careful documentation has been done, reconciliation is, for all practical purposes, impossible.

When management receives the conflicting reports, it is forced to make decisions based on politics and personalities because neither source is more or less credible. This is an example of the crisis of data credibility in the naturally evolving architecture.

This crisis is widespread and predictable. Why? As depicted in Figure 1.3, there are five reasons:

- No time basis of data

- The algorithmic differential of data
- The levels of extraction
- The problem of external data
- No common source of data from the beginning

The first reason for the predictability of the crisis is that there is no time basis for the data. Figure 1.4 shows such a time discrepancy. One department has extracted its data for analysis on a Sunday evening, and the other department extracted on a Wednesday afternoon. Is there any reason to believe that analysis done on one sample of data taken on one day will be the same as the analysis for a sample of data taken on another day? Of course not! Data is always changing within the corporation. Any correlation between analyzed sets of data that are taken at different points in time is only coincidental.

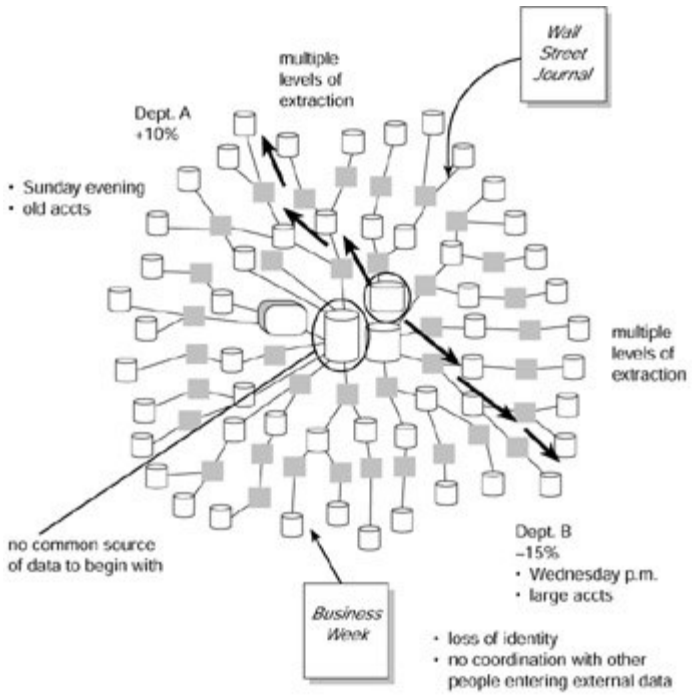

Figure 1.4: The reasons for the predictability of the crisis in data credibility in the naturally evolving architecture.

The second reason is the algorithmic differential. For example, one department has chosen to analyze all old accounts. Another department has chosen to analyze all large accounts. Is there any necessary correlation between the characteristics of customers who have old accounts and customers who have large accounts? Probably not. So why should a very different result surprise anyone?

The third reason is one that merely magnifies the first two reasons. Every time a new extraction is done, the probabilities of a discrepancy arise because of the timing or the algorithmic differential. And it is not unusual for a corporation to have eight or nine levels of extraction being done from the time the data enters the corporation's system to the time analysis is prepared for management. There are extracts, extracts of extracts, extracts of extracts of extracts, and so on. Each new level of extraction exaggerates the other problems that occur.

The fourth reason for the lack of credibility is the problem posed by external data. With today's technologies at the PC level, it is very easy to bring in data from outside sources. For

example, Figure 1.5 shows one analyst bringing data into the mainstream of analysis from the *Wall Street Journal*, and another analyst bringing data in from *Business Week*. However, when the analyst brings data in, he or she strips the identity of the external data. Because the origin of the data is not captured, it becomes generic data that could have come from any source.

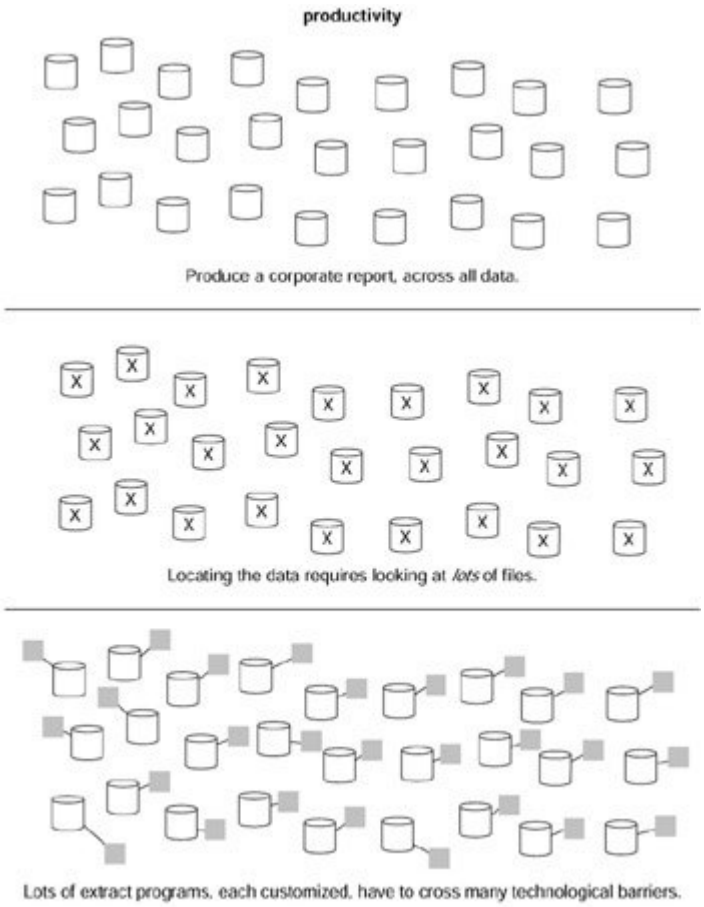

Figure 1.5: The naturally evolving architecture is not conducive to productivity.

Furthermore, the analyst who brings in data from the *Wall Street Journal* knows nothing about the data being entered from *Business Week*, and vice versa. No wonder, then, that external data contributes to the lack of credibility of data in the naturally evolving architecture.

The last contributing factor to the lack of credibility is that often there is no common source of data to begin with. Analysis for department A originates from file XYZ. Analysis for department B originates from database ABC. There is no synchronization or sharing of data whatsoever between file XYZ and database ABC.

Given these reasons, it is no small wonder that there is a crisis of credibility brewing in every organization that allows its legacy of hardware, software, and data to evolve naturally into the spider web.

Problems with Productivity

Data credibility is not the only major problem with the naturally evolving architecture. Productivity is also abysmal, especially when there is a need to analyze data across the organization.

Consider an organization that has been in business for a while and has built up a large collection of data, as shown in the top of Figure 1.5.

Management wants to produce a corporate report, using the many files and collections of data that have accumulated over the years. The designer assigned the task decides that three things must be done to produce the corporate report:

- Locate and analyze the data for the report.
- Compile the data for the report.
- Get programmer/analyst resources to accomplish these two tasks.

In order to locate the data, many files and layouts of data must be analyzed. Some files use Virtual Storage Access Method (VSAM), some use Information Management System (IMS), some use Adabas, some use Integrated Database Management System (IDMS). Different skill sets are required in order to access data across the enterprise. Furthermore, there are complicating factors: for example, two files might have an element called BALANCE, but the two elements are very different. In another case, one database might have a file known as CURRBAL, and another collection of data might have a file called INVLEVEL that happens to represent the same information as CURRBAL. Having to go through every piece of data—not just by name but by definition and calculation—is a very tedious process. But if the corporate report is to be produced, this exercise must be done properly. Unless data is analyzed and "rationalized," the report will end up mixing apples and oranges, creating yet another level of confusion.

The next task for producing the report is to compile the data once it is located. The program that must be written to get data from its many sources should be simple. It is complicated, though, by the following facts:

- Lots of programs have to be written.
- Each program must be customized.
- The programs cross every technology that the company uses.

In short, even though the report-generation program should be simple to write, retrieving the data for the report is tedious.

In a corporation facing exactly the problems described, an analyst recently estimated a very long time to accomplish the tasks, as shown in Figure 1.6.

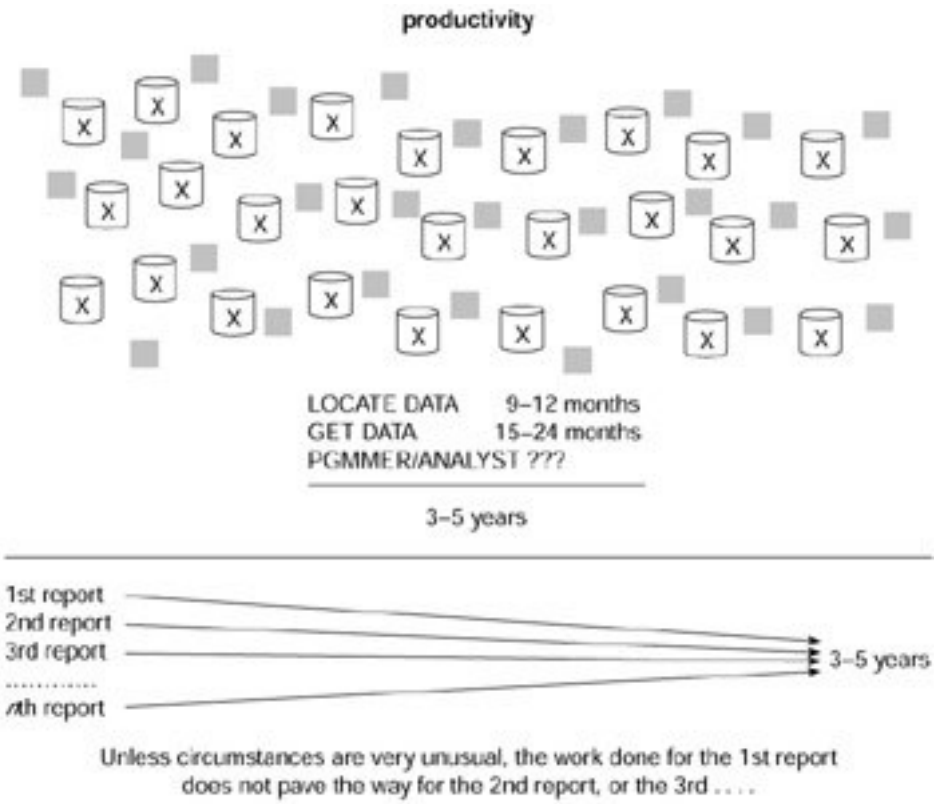

Figure 1.6: When the first report is being written, the requirements for future reports are not known.

If the designer had asked for only two or three man-months of resources, then generating the report might not have required much management attention. But when an analyst requisitions many resources, management must consider the request with all the other requests for resources and must prioritize the requests.

Creating the reports using a large amount of resources wouldn't be bad if there were a one-time penalty to be paid. In other words, if the first corporate report generated required a large amount of resources, and if all succeeding reports could build on the first report, then it might be worthwhile to pay the price for generating the first report. But that is not the case.

Unless future corporate reporting requirements are known in advance and are factored into building the first corporate report, each new corporate report will probably require the same large overhead! In other words, it is unlikely that the first corporate report will be adequate for future corporate reporting requirements.

Productivity, then, in the corporate environment is a major issue in the face of the naturally evolving architecture and its legacy systems. Simply stated, when using the spider web of legacy systems, information is expensive to access and takes a long time to create.

From Data to Information

As if productivity and credibility were not problems enough, there is another major fault of the naturally evolving architecture—the inability to go from data to information. At first glance, the notion of going from data to information seems to be an ethereal concept with little substance. But that is not the case at all.

Consider the following request for information, typical in a banking environment: “How has account activity differed this year from each of the past five years?”

Figure 1.7 shows the request for information.

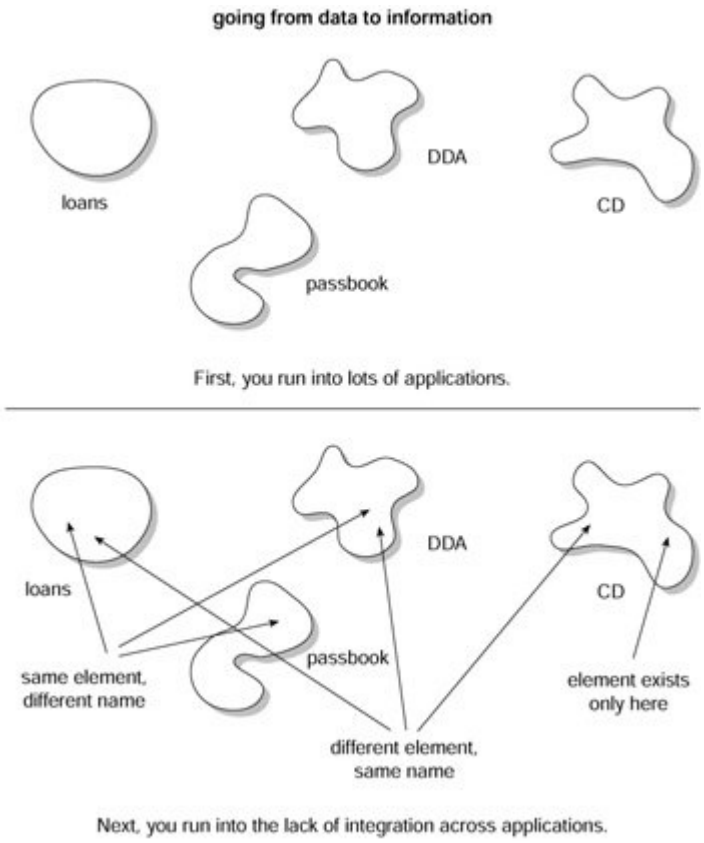

Figure 1.7: “How has account activity been different this year from each of the past five years for the financial institution?”

The first thing the DSS analyst discovers in trying to satisfy the request for information is that going to existing systems for the necessary data is the worst thing to do. The DSS analyst will have to deal with lots of unintegrated legacy applications. For example, a bank may have separate savings, loan, direct-deposit, and trust applications. However, trying to draw information from them on a regular basis is nearly impossible because the applications were never constructed with integration in mind, and they are no easier for the DSS analyst to decipher than they are for anyone else.

But integration is not the only difficulty the analyst meets in trying to satisfy an informational request. A second major obstacle is that there is not enough historical data stored in the applications to meet the needs of the DSS request.

Figure 1.8 shows that the loan department has up to two years’ worth of data. Passbook processing has up to one year of data. DDA applications have up to 60 days of data. And CD processing has up to 18 months of data. The applications were built to service the needs of current balance processing. They were never designed to hold the historical data needed for DSS analysis. It is no wonder, then, that going to existing systems for DSS analysis is a poor choice. But where else is there to go?

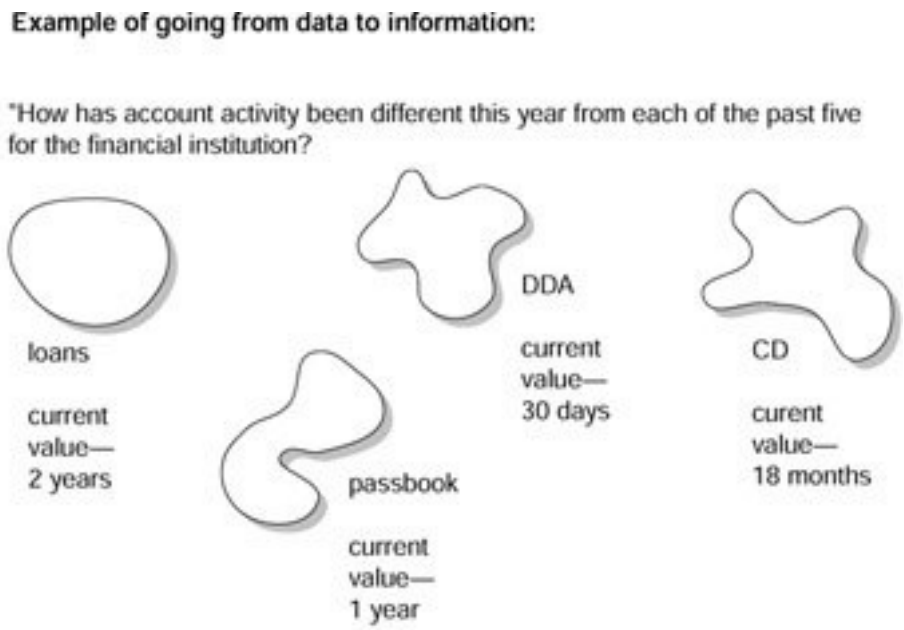

Figure 1.8: Existing applications simply do not have the historical data required to convert data into information.

The systems found in the naturally evolving architecture are simply inadequate for supporting information needs. They lack integration and there is a discrepancy between the time horizon (or parameter of time) needed for analytical processing and the available time horizon that exists in the applications.

A Change in Approach

The status quo of the naturally evolving architecture, where most shops began, simply is not robust enough to meet the future needs. What is needed is something much larger—a change in architectures. That is where the architected data warehouse comes in.

There are fundamentally two kinds of data at the heart of an “architected” environment—primitive data and derived data. Figure 1.9 shows some of the major differences between primitive and derived data.

Figure 1.9: The whole notion of data is changed in the architected environment.

Following are some other differences between the two.

- Primitive data is detailed data used to run the day-to-day operations of the company. Derived data has been summarized or otherwise calculated to meet the needs of the management of the company.
- Primitive data can be updated. Derived data can be recalculated but cannot be directly updated.
- Primitive data is primarily current-value data. Derived data is often historical data.
- Primitive data is operated on by repetitive procedures. Derived data is operated on by heuristic, nonrepetitive programs and procedures.
- Operational data is primitive; DSS data is derived.
- Primitive data supports the clerical function. Derived data supports the managerial function.

It is a wonder that the information processing community ever thought that both primitive and derived data would fit and peacefully coexist in a single database. In fact, primitive data and derived data are so different that they do not reside in the same database or even the same environment.

The Architected Environment

The natural extension of the split in data caused by the difference between primitive and derived data is shown in Figure 1.10.

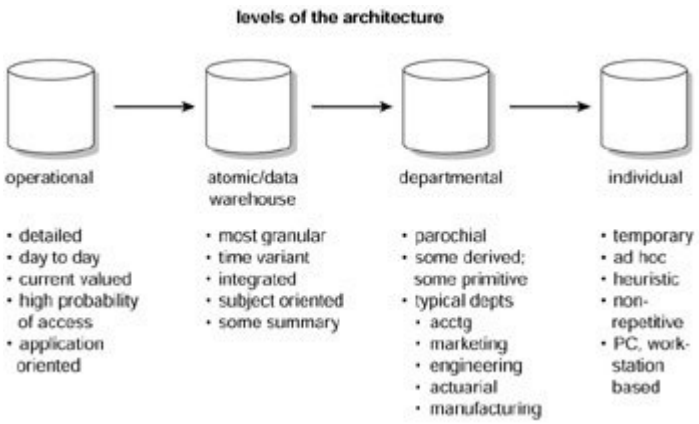

Figure 1.10: Although it is not apparent at first glance, there is very little redundancy of data across the architected environment.

There are four levels of data in the architected environment—the operational level, the atomic or the data warehouse level, the departmental (or the data mart level), and the individual level. These different levels of data are the basis of a larger architecture called the corporate information factory. The operational level of data holds application-oriented primitive data only and primarily serves the high-performance transaction-processing community. The data-warehouse level of data holds integrated, historical primitive data that cannot be updated. In addition, some derived data is found there. The departmental/data mart level of data contains derived data almost exclusively. The departmental/data mart level of data is shaped by end-user requirements into a form specifically suited to the needs of the department. And the individual level of data is where much heuristic analysis is done.

The different levels of data form a higher set of architectural entities. These entities constitute the corporate information factory, and they are described in more detail in my book *The Corporate Information Factory, Third Edition* (Wiley, 2002).

Some people believe the architected environment generates too much redundant data. Though it is not obvious at first glance, this is not the case at all. Instead, it is the spider web environment that generates the gross amounts of data redundancy.

Consider the simple example of data throughout the architecture, shown in Figure 1.11. At the operational level there is a record for a customer, J Jones. The operational-level record contains current-value data that can be updated at a moment's notice and shows the customer's current status. Of course, if the information for J Jones changes, the operational-level record will be changed to reflect the correct data.

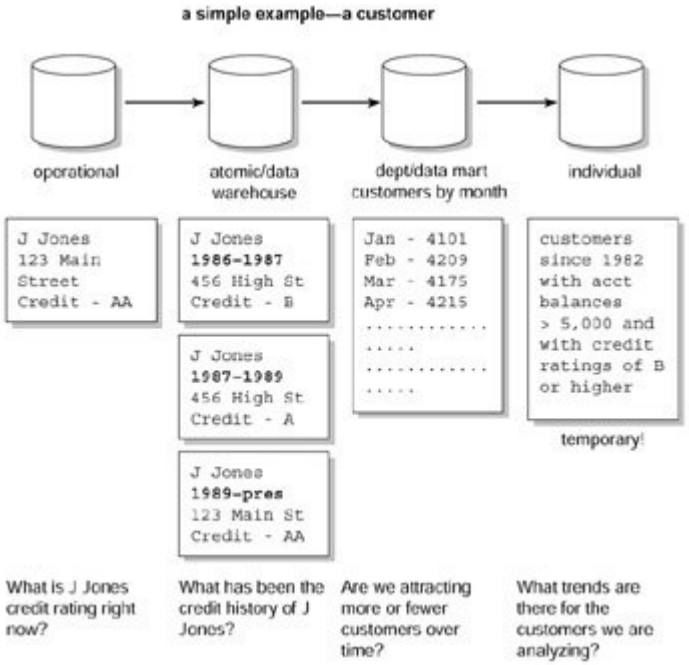

Figure 1.11: The kinds of queries for which the different levels of data can be used.

The data warehouse environment contains several records for J Jones, which show the history of information about J Jones. For example, the data warehouse would be searched to discover where J Jones lived last year. There is no overlap between the records in the operational environment, where current information is found, and the data warehouse environment, where historical information is found. If there is a change of address for J Jones, then a new record will be created in the data warehouse, reflecting the from and to dates that J Jones lived at the previous address. Note that the records in the data warehouse do not overlap. Also note that there is some element of time associated with each record in the data warehouse.

The departmental environment—sometimes called the data mart level, the OLAP level, or the multidimensional DBMS level—contains information useful to the different parochial departments of a company. There is a marketing departmental database, an accounting departmental database, an actuarial departmental database, and so forth. The data warehouse is the source of all departmental data. While data in the data mart certainly relates to data found in the operational level or the data warehouse, the data found in the departmental/data mart environment is fundamentally different from the data found in the data mart environment, because data mart data is denormalized, summarized, and shaped by the operating requirements of a single department**.**

Typical of data at the departmental/data mart level is a monthly customer file. In the file is a list of all customers by category. J Jones is tallied into this summary each month, along with many other customers. It is a stretch to consider the tallying of information to be redundant.

The final level of data is the individual level. Individual data is usually temporary and small. Much heuristic analysis is done at the individual level. As a rule, the individual levels of data are supported by the PC. Executive information systems (EIS) processing typically runs on the individual levels.

Data Integration in the Architected Environment

One important aspect of the architected environment that is not shown in Figure 1.11 is the integration of data that occurs across the architecture. As data passes from the operational environment to the data warehouse environment, it is integrated, as shown in Figure 1.12.

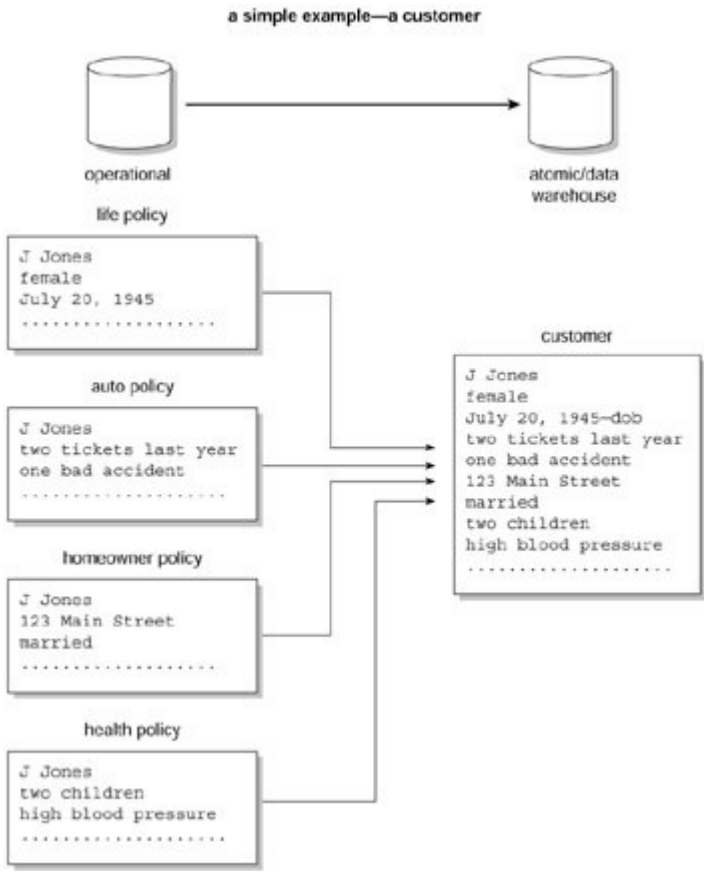

Figure 1.12: As data is transformed from the operational environment to the data warehouse environment, it is also integrated.

There is no point in bringing data over from the operational environment into the data warehouse environment without integrating it. If the data arrives at the data warehouse in an unintegrated state, it cannot be used to support a corporate view of data. And a corporate view of data is one of the essences of the architected environment.

In every environment the unintegrated operational data is complex and difficult to deal with. This is simply a fact of life. And the task of getting one's hands dirty with the process of integration is never pleasant. In order to achieve the real benefits of a data warehouse, though, it is necessary to undergo this painful, complex, and time-consuming exercise. Extract/transform/load (ETL) software can automate much of this tedious process. In addition, this process of integration has to be done only once. But, in any case, it is mandatory that data flowing into the data warehouse be integrated, not merely tossed—whole cloth—into the data warehouse from the operational environment.

Who Is the User?

Much about the data warehouse is fundamentally different from the operational environment. When developers and designers who have spent their entire careers in the operational environment first encounter the data warehouse, they often feel ill at ease. To help them appreciate why there is such a difference from the world they have known, they should understand a little bit about the different users of the data warehouse.

The data-warehouse user—also called the DSS analyst—is a businessperson first and foremost, and a technician second. The primary job of the DSS analyst is to define and discover information used in corporate decision-making.

It is important to peer inside the head of the DSS analyst and view how he or she perceives the use of the data warehouse. The DSS analyst has a mindset of "Give me what I say I want, then I can tell you what I really want." In other words, the DSS analyst operates in a mode of discovery. Only on seeing a report or seeing a screen can the DSS analyst begin to explore the possibilities for DSS. The DSS analyst often says, "Ah! Now that I see what the possibilities are, I can tell you what I really want to see. But until I know what the possibilities are I cannot describe to you what I want."

The attitude of the DSS analyst is important for the following reasons:

- It is legitimate. This is simply how DSS analysts think and how they conduct their business.
- It is pervasive. DSS analysts around the world think like this.
- It has a profound effect on the way the data warehouse is developed and on how systems using the data warehouse are developed.

The classical system development life cycle (SDLC) does not work in the world of the DSS analyst. The SDLC assumes that requirements are known at the start of design (or at least can be discovered). In the world of the DSS analyst, though, new requirements usually are the last thing to be discovered in the DSS development life cycle. The DSS analyst starts with existing requirements, but factoring in new requirements is almost an impossibility. A very different development life cycle is associated with the data warehouse.

The Development Life Cycle

We have seen how operational data is usually application oriented and as a consequence is unintegrated, whereas data warehouse data must be integrated. Other major differences also exist between the operational level of data and processing and the data warehouse level of data and processing. The underlying development life cycles of these systems can be a profound concern, as shown in Figure 1.13.

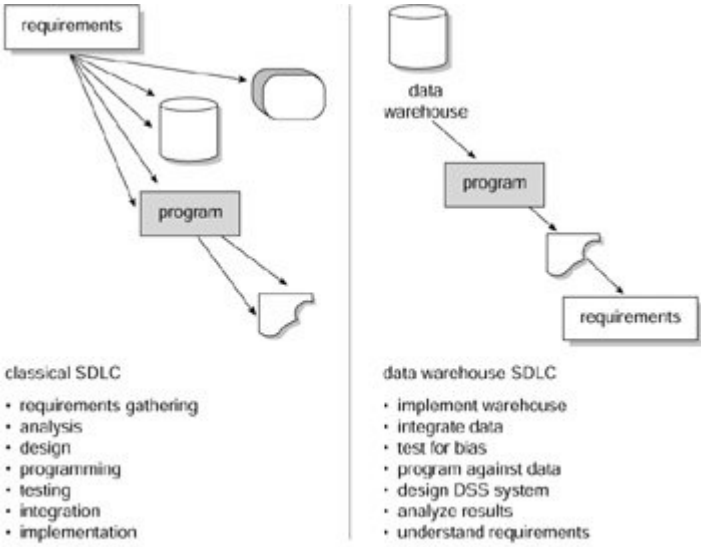

Figure 1.13: The system development life cycle for the data warehouse environment is almost exactly the opposite of the classical SDLC.

Figure 1.13 shows that the operational environment is supported by the classical systems development life cycle (the SDLC). The SDLC is often called the "waterfall" development approach because the different activities are specified and one activity-upon its completion-spills down into the next activity and triggers its start.

The development of the data warehouse operates under a very different life cycle, sometimes called the CLDS (the reverse of the SDLC). The classical SDLC is driven by requirements. In order to build systems, you must first understand the requirements. Then you go into stages of design and development. The CLDS is almost exactly the reverse: The CLDS starts with data. Once the data is in hand, it is integrated and then tested to see what bias there is to the data, if any. Programs are then written against the data. The results of the programs are analyzed, and finally the requirements of the system are understood. The CLDS is usually called a "spiral" development methodology. A spiral development methodology is included on the Web site, *www.billinmon.com*.

The CLDS is a classic data-driven development life cycle, while the SDLC is a classic requirements-driven development life cycle. Trying to apply inappropriate tools and techniques of development results only in waste and confusion. For example, the CASE world is dominated by requirements-driven analysis. Trying to apply CASE tools and techniques to the world of the data warehouse is not advisable, and vice versa.

Patterns of Hardware Utilization

Yet another major difference between the operational and the data warehouse environments is the pattern of hardware utilization that occurs in each environment. Figure 1.14 illustrates this.

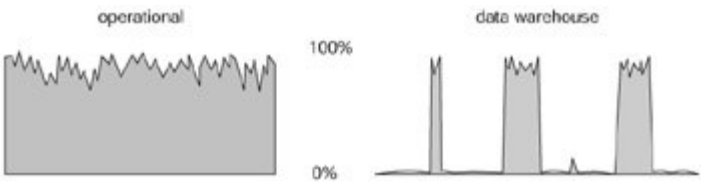

Figure 1.14: The different patterns of hardware utilization in the different environments.

The left side of Figure 1.14 shows the classic pattern of hardware utilization for operational processing. There are peaks and valleys in operational processing, but ultimately there is a relatively static and predictable pattern of hardware utilization.

There is an essentially different pattern of hardware utilization in the data warehouse environment (shown on the right side of the figure)—a binary pattern of utilization. Either the hardware is being utilized fully or not at all. It is not useful to calculate a mean percentage of utilization for the data warehouse environment. Even calculating the moments when the data warehouse is heavily used is not particularly useful or enlightening.

This fundamental difference is one more reason why trying to mix the two environments on the same machine at the same time does not work. You can optimize your machine either for operational processing or for data warehouse processing, but you cannot do both at the same time on the same piece of equipment.

Setting the Stage for Reengineering

Although indirect, there is a very beneficial side effect of going from the production environment to the architected, data warehouse environment. Figure 1.15 shows the progression.

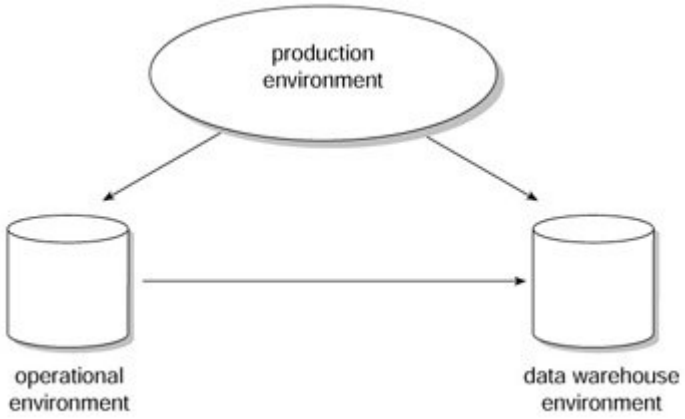

Figure 1.15: The transformation from the legacy systems environment to the architected, data warehouse-centered environment.

In Figure 1.15, a transformation is made in the production environment. The first effect is the removal of the bulk of data—mostly archival—from the production environment. The removal of massive volumes of data has a beneficial effect in various ways. The production environment is easer to:

- Correct
- Restructure
- Monitor
- Index

In short, the mere removal of a significant volume of data makes the production environment a much more malleable one.

Another important effect of the separation of the operational and the data warehouse environments is the removal of informational processing from the

production environment. Informational processing occurs in the form of reports, screens, extracts, and so forth. The very nature of information processing is constant change. Business conditions change, the organization changes, management changes, accounting practices change, and so on. Each of these changes has an effect on summary and informational processing. When informational processing is included in the production, legacy environment, maintenance seems to be eternal. But much of what is called maintenance in the production environment is actually informational processing going through the normal cycle of changes. By moving most informational processing off to the data warehouse, the maintenance burden in the production environment is greatly alleviated. Figure 1.16 shows the effect of removing volumes of data and informational processing from the production environment.

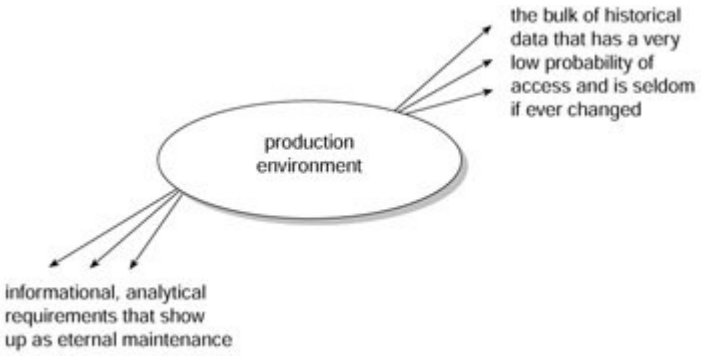

Figure 1.16: Removing unneeded data and information requirements from the production environment—the effects of going to the data warehouse environment.

Once the production environment undergoes the changes associated with transformation to the data warehouse-centered, architected environment, the production environment is primed for reengineering because:

- It is smaller.
- It is simpler.
- It is focused.

In summary, the single most important step a company can take to make its efforts in reengineering successful is to first go to the data warehouse environment.

Monitoring the Data Warehouse Environment

Once the data warehouse is built, it must be maintained. A major component of maintaining the data warehouse is managing performance, which begins by monitoring the data warehouse environment.

Two operating components are monitored on a regular basis: the data residing in the data warehouse and the usage of the data. Monitoring the data in the data warehouse environment is essential to effectively manage the data warehouse. Some of the important results that are achieved by monitoring this data include the following:

- Identifying what growth is occurring, where the growth is occurring, and at what rate the growth is occurring

- Identifying what data is being used
- Calculating what response time the end user is getting
- Determining who is actually using the data warehouse
- Specifying how much of the data warehouse end users are using
- Pinpointing when the data warehouse is being used
- Recognizing how much of the data warehouse is being used
- Examining the level of usage of the data warehouse

If the data architect does not know the answer to these questions, he or she can't effectively manage the data warehouse environment on an ongoing basis.

As an example of the usefulness of monitoring the data warehouse, consider the importance of knowing what data is being used inside the data warehouse. The nature of a data warehouse is constant growth. History is constantly being added to the warehouse. Summarizations are constantly being added. New extract streams are being created. And the storage and processing technology on which the data warehouse resides can be expensive. At some point the question arises, "Why is all of this data being accumulated? Is there really anyone using all of this?" Whether there is any legitimate user of the data warehouse, there certainly is a growing cost to the data warehouse as data is put into it during its normal operation.

As long as the data architect has no way to monitor usage of the data inside the warehouse, there is no choice but to continually buy new computer resources-more storage, more processors, and so forth. When the data architect can monitor activity and usage in the data warehouse, he or she can determine which data is not being used. It is then possible, and sensible, to move unused data to less expensive media. This is a very real and immediate payback to monitoring data and activity.

The data profiles that can be created during the data-monitoring process include the following:

- A catalog of all tables in the warehouse
- A profile of the contents of those tables
- A profile of the growth of the tables in the data warehouse
- A catalog of the indexes available for entry to the tables
- A catalog of the summary tables and the sources for the summary

The need to monitor activity in the data warehouse is illustrated by the following questions:

- What data is being accessed?
  - When?
  - By whom?
  - How frequently?
  - At what level of detail?
- What is the response time for the request?
- At what point in the day is the request submitted?
- How big was the request?
- Was the request terminated, or did it end naturally?

Response time in the DSS environment is quite different from response time in the online transaction processing (OLTP) environment. In the OLTP environment, response time is almost always mission critical. The business starts to suffer immediately when response time turns bad in OLTP. In the DSS environment there is no such relationship. Response time in the DSS data warehouse environment is always relaxed. There is no mission-critical nature to response time in DSS. Accordingly, response time in the DSS data warehouse environment is measured in minutes and hours and, in some cases, in terms of days.

Just because response time is relaxed in the DSS data warehouse environment does not mean that response time is not important. In the DSS data warehouse environment, the end user does development iteratively. This means that the next level of investigation of any iterative development depends on the results attained by the current analysis. If the end user does an iterative analysis and the turnaround time is only 10 minutes, he or she will be much more productive than if turnaround time is 24 hours. There is, then, a very important relationship between response time and productivity in the DSS environment. Just because response time in the DSS environment is not mission critical does not mean that it is not important.

The ability to measure response time in the DSS environment is the first step toward being able to manage it. For this reason alone, monitoring DSS activity is an important procedure.

One of the issues of response time measurement in the DSS environment is the question, "What is being measured?" In an OLTP environment, it is clear what is being measured. A request is sent, serviced, and returned to the end user. In the OLTP environment the measurement of response time is from the moment of submission to the moment of return. But the DSS data warehouse environment varies from the OLTP environment in that there is no clear time for measuring the return of data. In the DSS data warehouse environment often a lot of data is returned as a result of a query. Some of the data is returned at one moment, and other data is returned later. Defining the moment of return of data for the data warehouse environment is no easy matter. One interpretation is the moment of the first return of data; another interpretation is the last return of data. And there are many other possibilities for the measurement of response time; the DSS data warehouse activity monitor must be able to provide many different interpretations.

One of the fundamental issues of using a monitor on the data warehouse environment is where to do the monitoring. One place the monitoring can be done is at the end-user terminal, which is convenient many machine cycles are free here and the impact on systemwide performance is minimal. To monitor the system at the end-user terminal level implies that each terminal that will be monitored will require its own administration. In a world where there are as many as 10,000 terminals in a single DSS network, trying to administer the monitoring of each terminal is nearly impossible.

The alternative is to do the monitoring of the DSS system at the server level. After the query has been formulated and passed to the server that manages the data warehouse, the monitoring of activity can occur. Undoubtedly, administration of the monitor is much easier here. But there is a very good possibility that a systemwide performance penalty will be incurred. Because the monitor is using resources at the server, the impact on performance is felt throughout the DSS data warehouse environment. The placement of the monitor is an important issue that must be thought out carefully. The trade-off is between ease of administration and minimization of performance requirements.

One of the most powerful uses of a monitor is to be able to compare today's results against an "average" day. When unusual system conditions occur, it is often useful to ask, "How different is today from the average day?" In many cases, it will be seen that the variations in performance are not nearly as bad as imagined. But in order to make such a comparison, there needs to be an average-day profile, which contains the standard important measures that describe a day in the DSS environment. Once the current day is measured, it can then be compared to the average-day profile.

Of course, the average day changes over time, and it makes sense to track these changes periodically so that long-term system trends can be measured.

Summary

This chapter has discussed the origins of the data warehouse and the larger architecture into which the data warehouse fits. The architecture has evolved throughout the history of the different stages of information processing. There are four levels of data and processing in the architecture—the operational level, the data warehouse level, the departmental/data mart level, and the individual level.

The data warehouse is built from the application data found in the operational environment. The application data is integrated as it passes into the data warehouse. The act of integrating data is always a complex and tedious task. Data flows from the data warehouse into the departmental/data mart environment. Data in the departmental/data mart environment is shaped by the unique processing requirements of the department.

The data warehouse is developed under a completely different development approach than that used for classical application systems. Classically applications have been developed by a life cycle known as the SDLC. The data warehouse is developed under an approach called the spiral development methodology. The spiral development approach mandates that small parts of the data warehouse be developed to completion, then other small parts of the warehouse be developed in an iterative approach.

The users of the data warehouse environment have a completely different approach to using the system. Unlike operational users who have a straightforward approach to defining their requirements, the data warehouse user operates in a mindset of discovery. The end user of the data warehouse says, “Give me what I say I want, then I can tell you what I really want.”

Chapter 2: The Data Warehouse Environment

Overview

The data warehouse is the heart of the architected environment, and is the foundation of all DSS processing. The job of the DSS analyst in the data warehouse environment is immeasurably easier than in the classical legacy environment because there is a single integrated source of data (the data warehouse) and because the granular data in the data warehouse is easily accessible.

This chapter will describe some of the more important aspects of the data warehouse. A data warehouse is a subject-oriented, integrated, nonvolatile, and time-variant collection of data in support of management's decisions. The data warehouse contains granular corporate data.

The subject orientation of the data warehouse is shown in Figure 2.1. Classical operations systems are organized around the applications of the company. For an insurance company, the applications may be auto, health, life, and casualty. The major subject areas of the insurance corporation might be customer, policy, premium, and claim. For a manufacturer, the major subject areas might be product, order, vendor, bill of material, and raw goods. For a retailer, the major subject areas may be product, SKU, sale, vendor, and so forth. Each type of company has its own unique set of subjects.

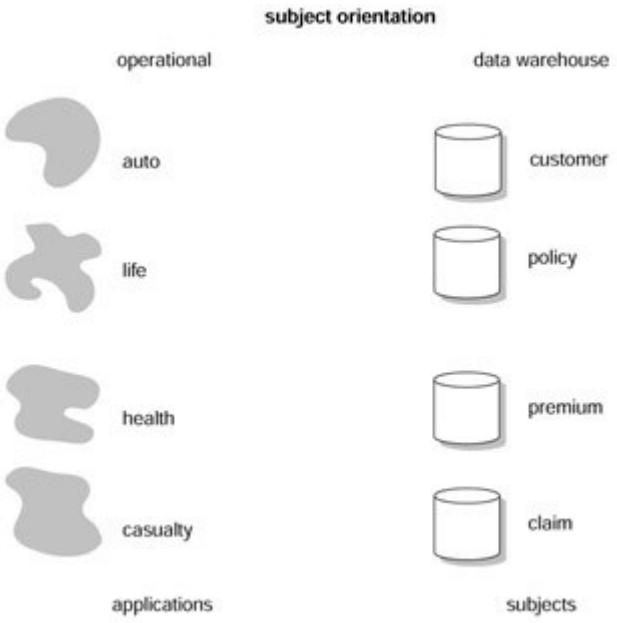

Figure 2.1: An example of a subject orientation of data.

The second salient characteristic of the data warehouse is that it is integrated. Of all the aspects of a data warehouse, integration is the most important. Data is fed from multiple disparate sources into the data warehouse. As the data is fed it is converted, reformatted, resequenced, summarized, and so forth. The result is that data—once it resides in the data warehouse—has a single physical corporate image. Figure 2.2 illustrates the integration that occurs when data passes from the application-oriented operational environment to the data warehouse.

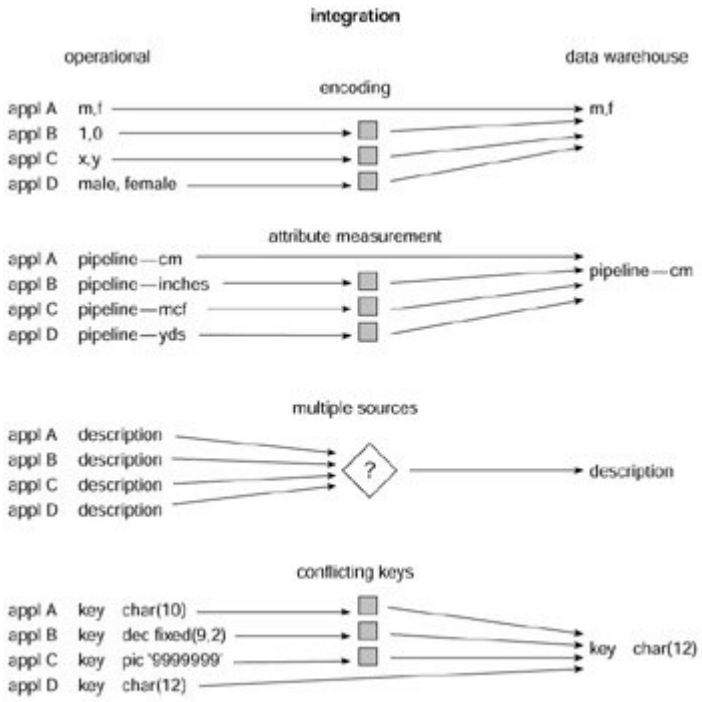

Figure 2.2: The issue of integration.

Design decisions made by applications designers over the years show up in different ways. In the past, when application designers built an application, they never considered that the data they were operating on would ever have to be integrated with other data. Such a consideration was only a wild theory. Consequently, across multiple applications there is no application consistency in encoding, naming conventions, physical attributes, measurement of attributes, and so forth. Each application designer has had free rein to make his or her own design decisions. The result is that any application is very different from any other application.

Data is entered into the data warehouse in such a way that the many inconsistencies at the application level are undone. For example, in Figure 2.2, as far as encoding of gender is concerned, it matters little whether data in the warehouse is encoded as m/f or 1/0 . What does matter is that regardless of method or source application, warehouse encoding is done consistently. If application data is encoded as X/Y, it is converted as it is moved to the warehouse. The same consideration of consistency applies to all application design issues, such as naming conventions, key structure, measurement of attributes, and physical characteristics of data.

The third important characteristic of a data warehouse is that it is nonvolatile. Figure 2.3 illustrates nonvolatility of data and shows that operational data is regularly accessed and manipulated one record at a time. Data is updated in the operational environment as a regular matter of course, but data warehouse data exhibits a very different set of characteristics. Data warehouse data is loaded (usually en masse) and accessed, but it is not updated (in the general sense). Instead, when data in the data warehouse is loaded, it is loaded in a snapshot, static format. When subsequent changes occur, a new snapshot record is written. In doing so a history of data is kept in the data warehouse.

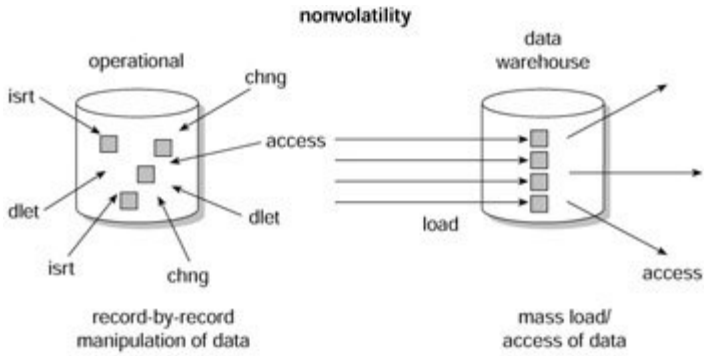

Figure 2.3: The issue of nonvolatility.

The last salient characteristic of the data warehouse is that it is time variant. Time variancy implies that every unit of data in the data warehouse is accurate as of some one moment in time. In some cases, a record is time stamped. In other cases, a record has a date of transaction. But in every case, there is some form of time marking to show the moment in time during which the record is accurate. Figure 2.4 illustrates how time variancy of data warehouse data can show up in several ways.

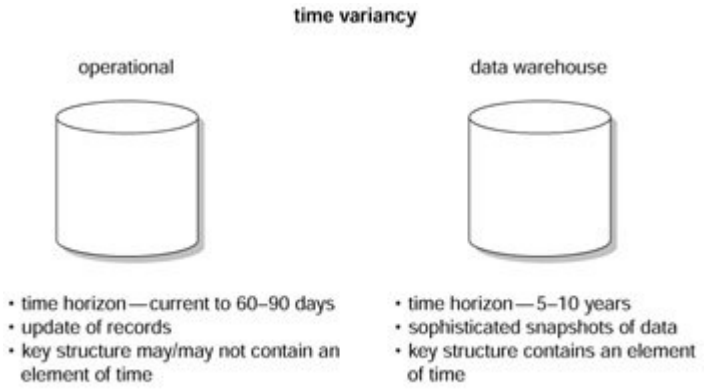

Figure 2.4: The issue of time variancy.

Different environments have different time horizons. A time horizon is the parameters of time represented in an environment. The collective time horizon for the data found inside a data warehouse is significantly longer than that of operational systems. A 60-to-90-day time horizon is normal for operational systems; a 5-to-10-year time horizon is normal for the data warehouse. As a result of this difference in time horizons, the data warehouse contains *much* more history than any other environment.

Operational databases contain current-value data-data whose accuracy is valid as of the moment of access. For example, a bank knows how much money a customer has on deposit at any moment in time. Or an insurance company knows what policies are in force at any moment in time. As such, current-value data can be updated as business conditions change. The bank balance is changed when the customer makes a deposit. The insurance coverage is changed when a customer lets a policy lapse. Data warehouse data is very unlike current-value data, however. Data warehouse data is nothing more than a sophisticated series of snapshots, each taken at one moment in time. The effect created by the series of snapshots is that the data warehouse has a historical sequence of activities and events, something not at all apparent in a current-value environment where only the most current value can be found.

The key structure of operational data may or may not contain some element of time, such as year, month, day, and so on. The key structure of the data warehouse always contains some element of time. The embedding of the element of time can take many forms, such as a time stamp on every record, a time stamp for a whole database, and so forth.

The Structure of the Data Warehouse

Figure 2.5 shows that there are different levels of detail in the data warehouse. There is an older level of detail (usually on alternate, bulk storage), a current level of detail, a level of lightly summarized data (the data mart level), and a level of highly summarized data. Data flows into the data warehouse from the operational environment. Usually significant transformation of data occurs at the passage from the operational level to the data warehouse level.

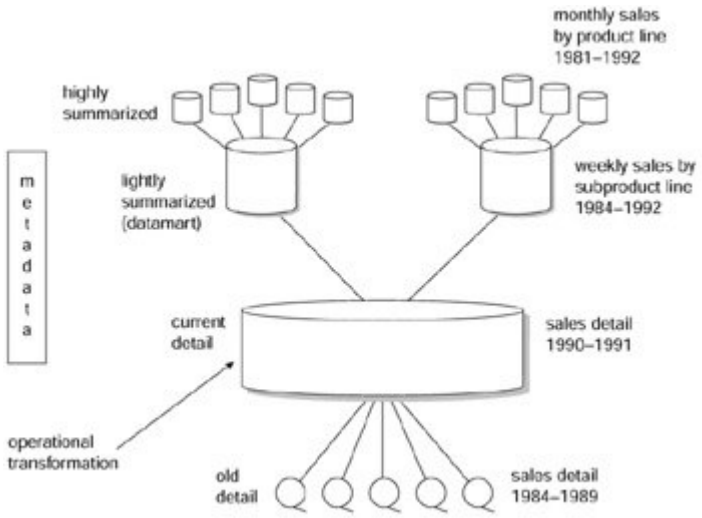

Figure 2.5: The structure of the data warehouse.

Once the data ages, it passes from current detail to older detail. As the data is summarized, it passes from current detail to lightly summarized data, then from lightly summarized data to highly summarized data.

Subject Orientation

The data warehouse is oriented to the major subject areas of the corporation that have been defined in the high-level corporate data model. Typical subject areas include the following:

- Customer
- Product
- Transaction or activity
- Policy
- Claim
- Account

Each major subject area is physically implemented as a series of related tables in the data warehouse. A subject area may consist of 10, 100, or even more physical tables that are all related. For example, the subject area implementation for a customer might look like that shown in Figure 2.6.

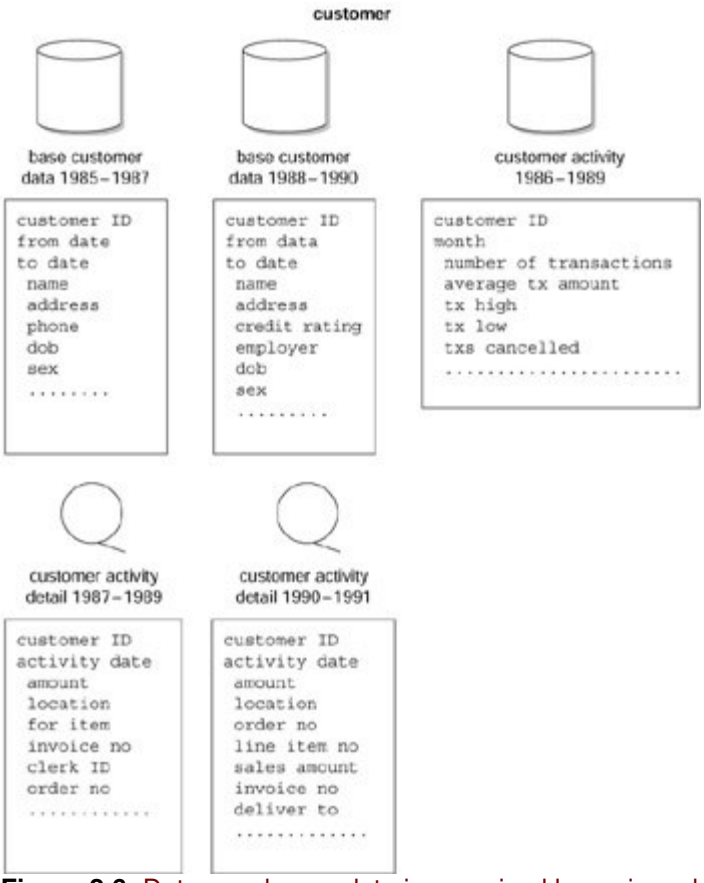

Figure 2.6: Data warehouse data is organized by major subject area—in this case, by customer.

There are five related physical tables in Figure 2.6, each of which has been designed to implement a part of a major subject area—customer. There is a base table for customer information as defined from 1985 to 1987. There is another for the definition of customer data between 1988 and 1990. There is a cumulative customer activity table for activities between 1986 and 1989. Each month a summary record is written for each customer record based on customer activity for the month.

There are detailed activity files by customer for 1987 through 1989 and another one for 1990 through 1991. The definition of the data in the files is different, based on the year.

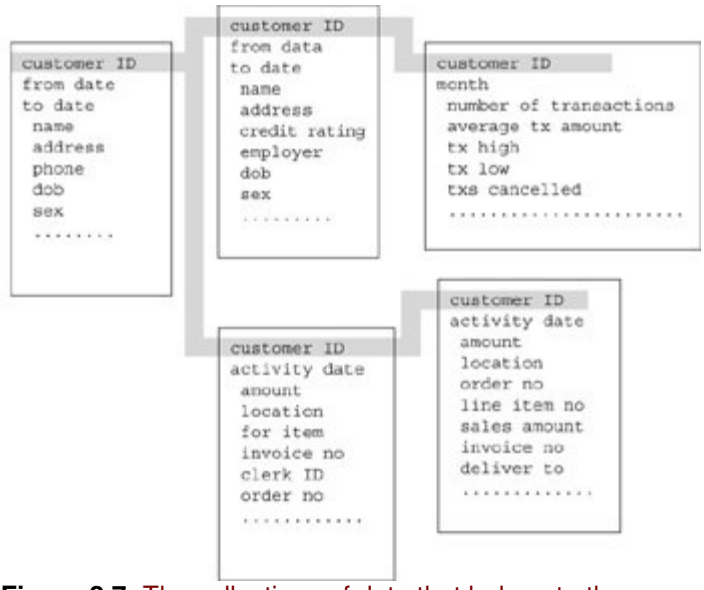

Figure 2.7: The collections of data that belong to the same subject area are tied together by a common key.

All of the physical tables for the customer subject area are related by a common key. Figure 2.7 shows that the key—customer ID—connects all of the data found in the customer subject area. Another interesting aspect of the customer subject area is that it may reside on different media, as shown in Figure 2.8. There is nothing to say that a physical table must reside on disk, even if it relates to other data that does reside on a disk.

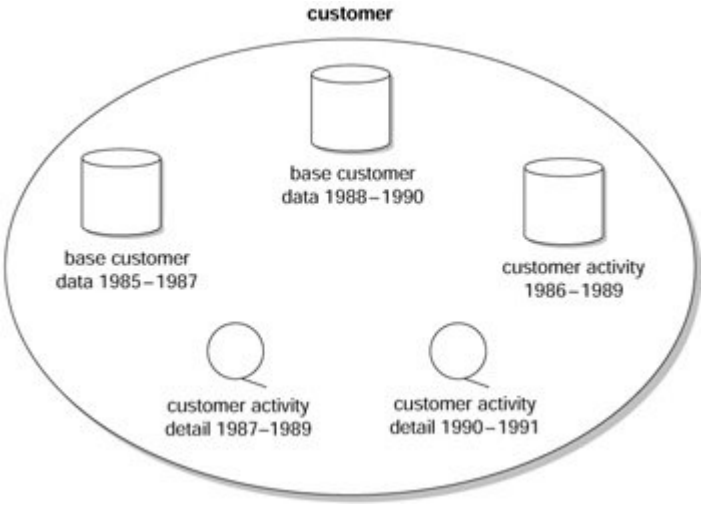

Figure 2.8: The subject area may contain data on different media in the data warehouse.

Figure 2.8 shows that some of the related subject area data resides on direct access storage device (DASD) and some resides on magnetic tape. One implication of data residing on different media is that there may be more than one DBMS managing the data in a warehouse or that some data may not be managed by a DBMS at all. Just because data resides on magnetic tape or some storage media other than disk storage does not mean that the data is not a part of the data warehouse.

Data that has a high probability of access and a low volume of storage resides on a medium that is fast and relatively expensive. Data that has a low probability of access and is bulky resides on a medium that is cheaper and slower to access. Usually (but not always) data that is older has a lower probability of access. As a rule, the older data resides on a medium other than disk storage.

DASD and magnetic tape are the two most popular media on which to store data in a data warehouse. But they are not the only media; two others that should not be overlooked are fiche and optical disk. Fiche is good for storing detailed records that never have to be reproduced in an electronic medium again. Legal records are often stored on fiche for an indefinite period of time. Optical disk storage is especially good for data warehouse storage because it is cheap, relatively fast, and able to hold a mass of data. Another reason why optical disk is useful is that data warehouse data, once written, is seldom, if ever, updated. This last characteristic makes optical disk storage a very desirable choice for data warehouses.

Another interesting aspect of the files (shown in Figure 2.8) is that there is both a level of summary and a level of detail for the same data. Activity by month is summarized. The detail that supports activity by month is stored at the magnetic tape level of data. This is a form of a "shift in granularity," which will be discussed later.

When data is organized around the subject-in this case, the customer—each key has an element of time, as shown in Figure 2.9.

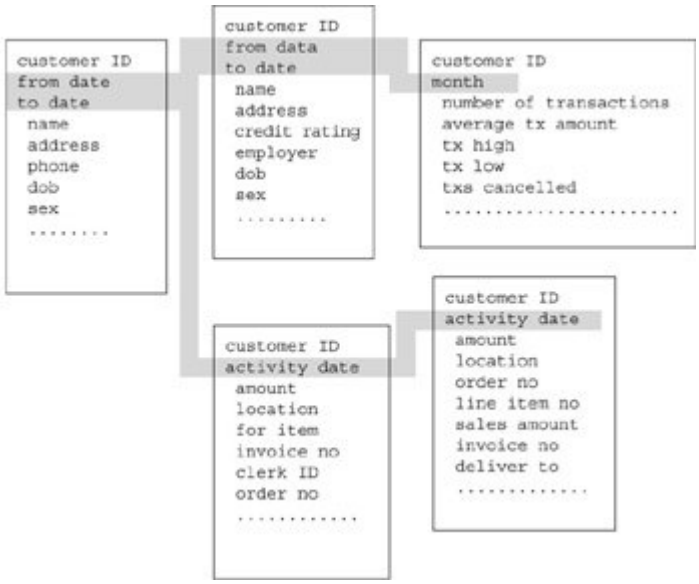

Figure 2.9: Each table in the data warehouse has an element of time as a part of the key structure, usually the lower part.

Some tables are organized on a from-date-to-date basis. This is called a continuous organization of data. Other tables are organized on a cumulative monthly basis, and others on an individual date of record or activity basis. But all records have some form of date attached to the key, usually the lower part of the key.

Day 1-Day *n* Phenomenon

Data warehouses are not built all at once. Instead, they are designed and populated a step at a time, and as such are evolutionary, not revolutionary. The costs of building a data

warehouse all at once, the resources required, and the disruption to the environment all dictate that the data warehouse be built in an orderly iterative, step-at-a-time fashion. The "big bang" approach to data warehouse development is simply an invitation to disaster and is never an appropriate alternative.

Figure 2.10 shows the typical process of building a data warehouse. On day 1 there is a polyglot of legacy systems essentially doing operational, transactional processing. On day 2, the first few tables of the first subject area of the data warehouse are populated. At this point, a certain amount of curiosity is raised, and the users start to discover data warehouses and analytical processing.

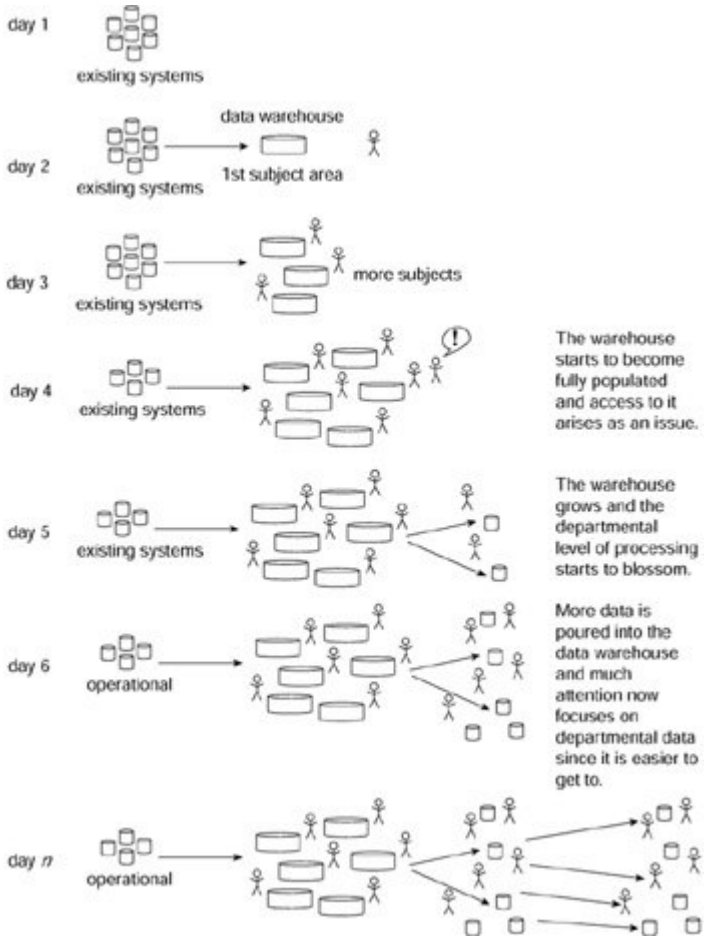

Figure 2.10: Day 1-day n phenomenon.

On day 3, more of the data warehouse is populated, and with the population of more data comes more users. Once users find there is an integrated source of data that is easy to get to and has a historical basis designed for looking at data over time, there is more than curiosity. At about this time, the serious DSS analyst becomes attracted to the data warehouse.

On day 4, as more of the warehouse becomes populated, some of the data that had resided in the operational environment becomes properly placed in the data warehouse. And the data warehouse is now discovered as a source for doing analytical processing. All sorts of DSS applications spring up. Indeed, so many users and so many requests for processing, coupled with a rather large volume of data that now resides in the warehouse, appear that

some users are put off by the effort required to get to the data warehouse. The competition to get at the warehouse becomes an obstacle to its usage.

On day 5, departmental databases (data mart or OLAP) start to blossom. Departments find that it is cheaper and easier to get their processing done by bringing data from the data warehouse into their own departmental processing environment. As data goes to the departmental level, a few DSS analysts are attracted.

On day 6, the land rush to departmental systems takes place. It is cheaper, faster, and easier to get departmental data than it is to get data from the data warehouse. Soon end users are weaned from the detail of data warehouse to departmental processing.

On day *n*, the architecture is fully developed. All that is left of the original set of production systems is operational processing. The warehouse is full of data. There are a few direct users of the data warehouse. There are a lot of departmental databases. Most of the DSS analytical processing occurs at the departmental level because it is easier and cheaper to get the data needed for processing there.

Of course, evolution from day 1 to day *n* takes a long time. The evolution does not happen in a matter of days. Several years is the norm. During the process of moving from day 1 to day *n* the DSS environment is up and functional.

Note that the spider web seems to have reappeared in a larger, more grandiose form. Such is not the case at all, although the explanation is rather complex. Refer to “The Cabinet Effect,” in the May 1991 edition of *Data Base Programming Design*, for an in-depth explanation of why the architected environment is not merely a recreation of the spider web environment.

The day 1-day *n* phenomenon described here is the ideal way to get to the data warehouse. There are many other paths. One such path is through the building of data marts first. This path is short sighted and leads to a great deal of waste.

Granularity

The single most important aspect of design of a data warehouse is the issue of granularity. Indeed, the issue of granularity permeates the entire architecture that surrounds the data warehouse environment. Granularity refers to the level of detail or summarization of the units of data in the data warehouse. The more detail there is, the lower the level of granularity. The less detail there is, the higher the level of granularity.

For example, a simple transaction would be at a low level of granularity. A summary of all transactions for the month would be at a high level of granularity.

Granularity of data has always been a major design issue. In early operational systems, granularity was taken for granted. When detailed data is being updated, it is almost a given that data be stored at the lowest level of granularity. In the data warehouse environment, though, granularity is not assumed. Figure 2.11 illustrates the issues of granularity.

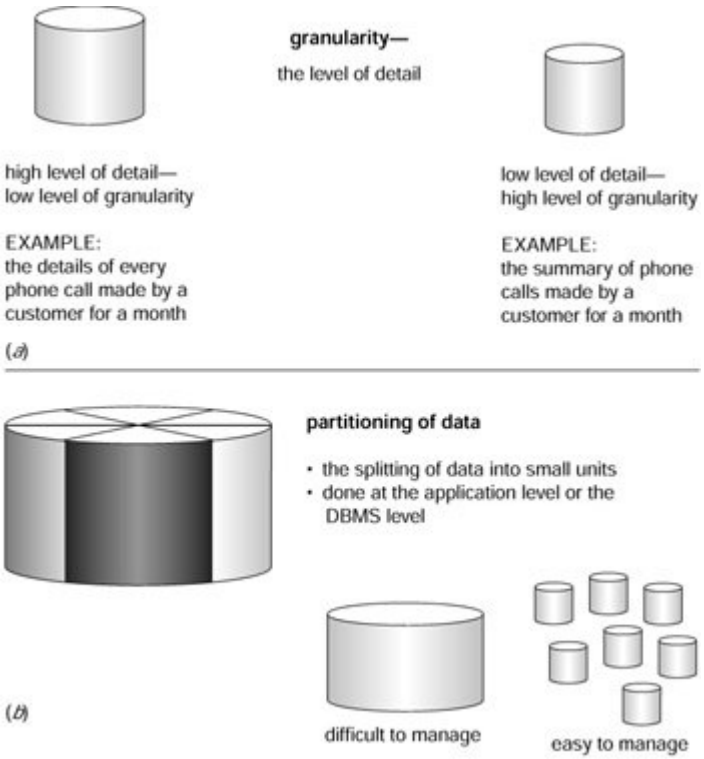

Figure 2.11: Major design issues of the data warehouse: granularity, partitioning, and proper design.

Granularity is the major design issue in the data warehouse environment because it profoundly affects the volume of data that resides in the data warehouse and the type of query that can be answered. The volume of data in a warehouse is traded off against the level of detail of a query.

In almost all cases, data comes into the data warehouse at too high a level of granularity. This means that the developer must spend a lot of resources breaking the data apart. Occasionally, though, data enters the warehouse at too low a level of granularity. An example of data at too low a level of granularity is the Web log data generated by the Web-based ebusiness environment. Web log clickstream data must be edited, filtered, and summarized before its granularity is fit for the data warehouse environment.

The Benefits of Granularity

Many organizations are surprised to find that data warehousing provides an invaluable foundation for many different types of DSS processing. Organizations may build a data warehouse for one purpose, but they discover that it can be used for many other kinds of DSS processing. Although infrastructure for the data warehouse is expensive and difficult to build, it has to be built only once. After the data warehouse has been properly constructed, it provides the organization with a foundation that is extremely flexible and reusable.

The granular data found in the data warehouse is the key to reusability, because it can be used by many people in different ways. For example, within a corporation, the same data might be used to satisfy the needs of marketing, sales, and accounting. All three departments look at the basic same data. Marketing may want to see sales on a monthly basis by geographic district, sales may want to see sales by salesperson by sales district on a weekly basis, and finance may want to see recognizable revenue on a quarterly basis by

product line. All of these types of information are closely related, yet slightly different. With a data warehouse, the different organizations are able to look at the data as they wish to see it.

Looking at the data in different ways is only one advantage of having a solid foundation. A related benefit is the ability to reconcile data, if needed. Once there is a single foundation on which everyone relies, if there is a need to explain a discrepancy in analyses between two or more departments, then reconciliation is relatively simple.

Another related benefit is flexibility. Suppose that marketing wishes to alter how it looks at data. Having a foundation in place allows this to be accomplished easily.

Another benefit of granular data is that it contains a history of activities and events across the corporation. And the level of granularity is detailed enough that the data can be reshaped across the corporation for many different needs.

But perhaps the largest benefit of a data warehouse foundation is that future unknown requirements can be accommodated. Suppose there is a new requirement to look at data, or the state legislature passes a new law, or OPEC changes its rules for oil allocation, or the stock market crashes. There is a constant stream of new requirements for information because change is inevitable. With the data warehouse in place, the corporation can easily respond to change. When a new requirement arises and there is a need for information, the data warehouse is already available for analysis, and the organization is prepared to handle the new requirements.

An Example of Granularity

Figure 2.12 shows an example of the issues of granularity. The left side shows a low level of granularity. Each activity—in this case, a phone call—is recorded in detail. At the end of the month, each customer has, on average, 200 records (one for each recorded phone call throughout the month) that require about 40,000 bytes collectively.

The right side of the figure shows a higher level of granularity. A high level of detail refers to a low level of granularity. A low level of detail refers to a high level of granularity. The data shown in Fig 2.12 is at a high level of granularity. It represents the summary information stet. Each record summarizes one month's activity for one customer, which requires about 200 bytes.

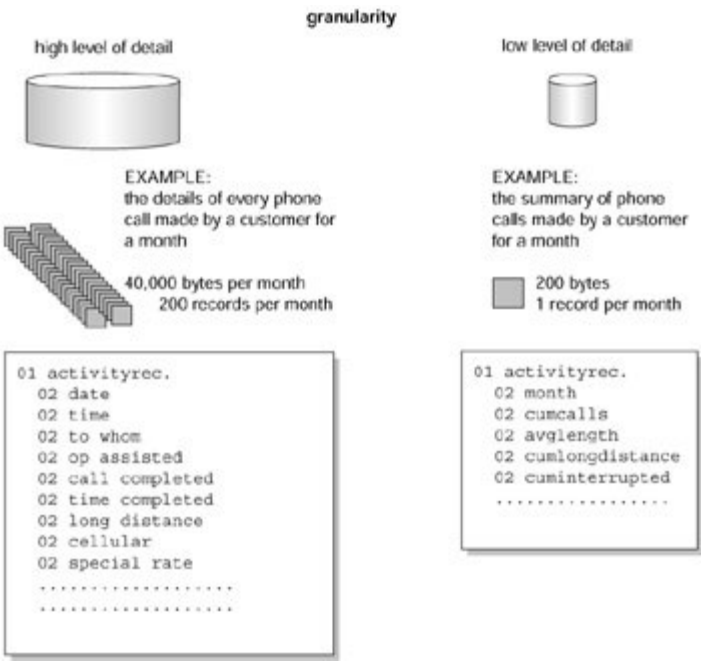

Figure 2.12: Determining the level of granularity is the most important design issue in the data warehouse environment.

It is obvious that if space is a problem in a data warehouse (and volume of data is always the first and major issue in the data warehouse), a high level of granularity is a much more efficient way of representing data than a representation using a low level of granularity.

Not only are many fewer bytes of data required with a higher level of granularity, but fewer index entries are needed. But the volume of data and the issue of raw space are not the only relevant issues. The amount of processing power that needs to be used in the face of a large volume of data in order to access the data is a factor as well.

There is, then, a very good case for the compaction of data in a data warehouse. When data is compacted, significant savings can be realized in the amount of DASD used, the number of index entries required, and the processor resources required to manipulate data.

Another aspect to the compaction of data occurs when the level of granularity is raised. Figure 2.13 demonstrates the trade-off. As the level of granularity of data rises, there is a corresponding loss in the ability to answer queries using the data. Put another way, with a very low level of granularity, you can answer practically any query. But a high level of granularity limits the number of questions that the data can handle.

Another consideration in designing granularity is determining which architectural entities will feed off the data warehouse. Each entity has its own unique considerations. The data warehouse must be designed to feed the lowest level of granularity needed by any architectural entity.

To illustrate the effect of granularity on the ability to do queries, in Figure 2.13, the following query is made: “Did Cass Squire call his girlfriend in Boston last week?”

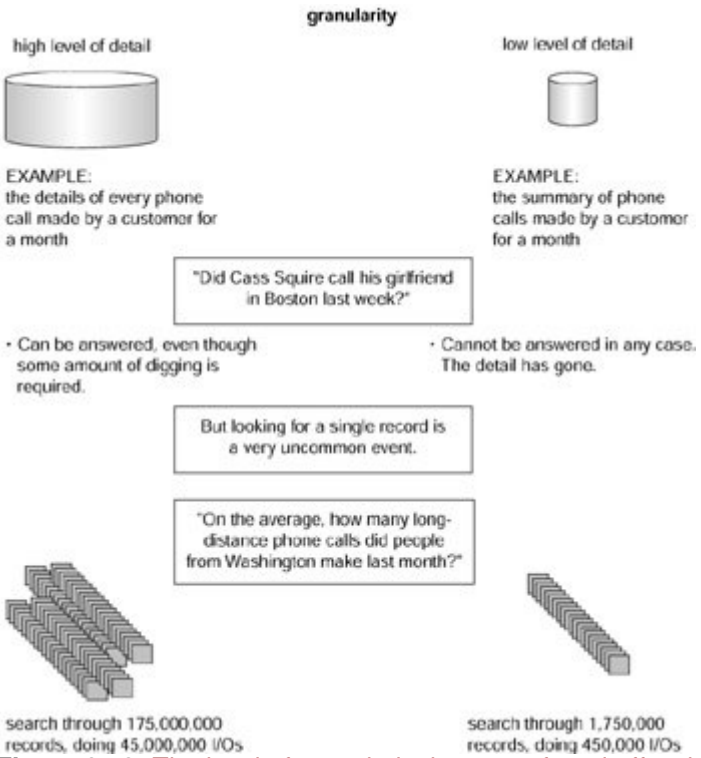

Figure 2.13: The level of granularity has a profound effect both on what questions can be answered and on what resources are required to answer a question.

With a low level of granularity, the query can be answered. It may take a lot of resources to thumb through a lot of records, but at the end of the day, whether Cass called his girlfriend in Boston last week can be determined.

But with a high level of detail, there is no way to definitively answer the question. If all that is kept about Cass Squire in the data warehouse is the total number of calls that he made for a given week or month, whether one of those calls went to Boston cannot be determined.

When doing DSS processing in the data warehouse environment, examining only a single event is rare. A collective view of data is much more common. To achieve a collective view requires looking at a large number of records.

For example, suppose the following collective query is made: “On the average, how many long-distance phone calls did people from Washington make last month?”

This type of query is normal in a DSS environment. Of course, it can be answered by both the high level and the low level of granularity. In answering it, though, there is a tremendous difference in the resources used. A low level of granularity requires sorting through each record, because many resources are required when using very detailed data.

But using the high level of granularity, the data is much more compact and can provide an answer relatively quickly. If it contains sufficient detail, data with a high level of granularity is much more efficient to use.

Figure 2.14 shows the trade-off in determining the level of data granularity to use. This trade-off must be considered very carefully at the outset of designing and constructing the data warehouse.

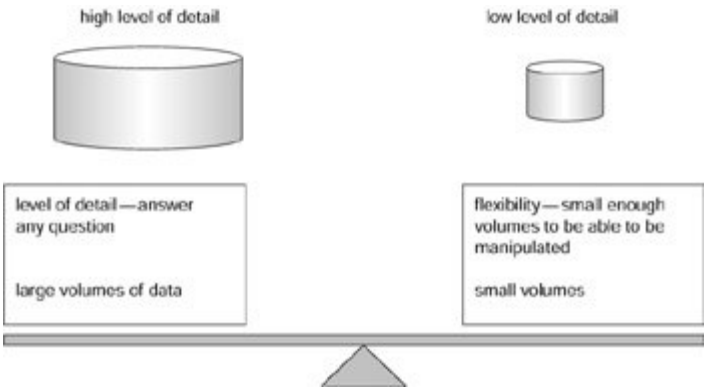

Figure 2.14: The trade-off with granularity is so stark that for most organizations the best solution is some form of multiple levels of granularity.

Dual Levels of Granularity

Most of the time, there is a great need for efficiency in storing and accessing data, and for the ability to analyze data in great detail. (In other words, the organization wants to have its cake and eat it, too!) When an organization has lots of data in the warehouse, it makes eminent sense to consider two (or more) levels of granularity in the detailed portion of the data warehouse. In fact, there is such a need for more than one level of granularity that a dual level of granularity design should be the default for almost every shop. Figure 2.15 shows two levels of granularity at the detailed level of the data warehouse.

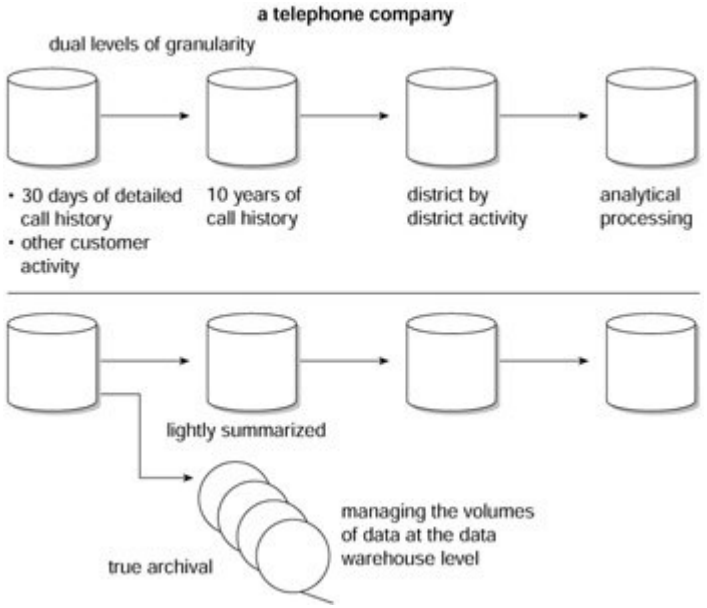

Figure 2.15: The volume of data is such that most organizations need to have two levels of granularity in the data warehouse.

Called a dual level of granularity, the design shown in Figure 2.15—a phone company—fits the needs of most shops. There is a tremendous amount of detail at the operational level. Most of this detail is needed for the billing systems. Up to 30 days of detail is stored in the operational level.

The data warehouse in this example contains two types of data-lightly summarized data and "true archival" detail data. The data in the data warehouse can go back 10 years. The data that emanates from the data warehouse is "district" data that flows to the different districts of the telephone company. Each district then analyzes its data independently from other districts. Much heuristic analytical processing occurs at the individual level.

A light summarization data is detailed data that has been summarized only to a very small extent. For example, phone call information may be summarized by the hour. Or bank checking information may be summarized by the day. Figure 2.16 shows such a summarization.

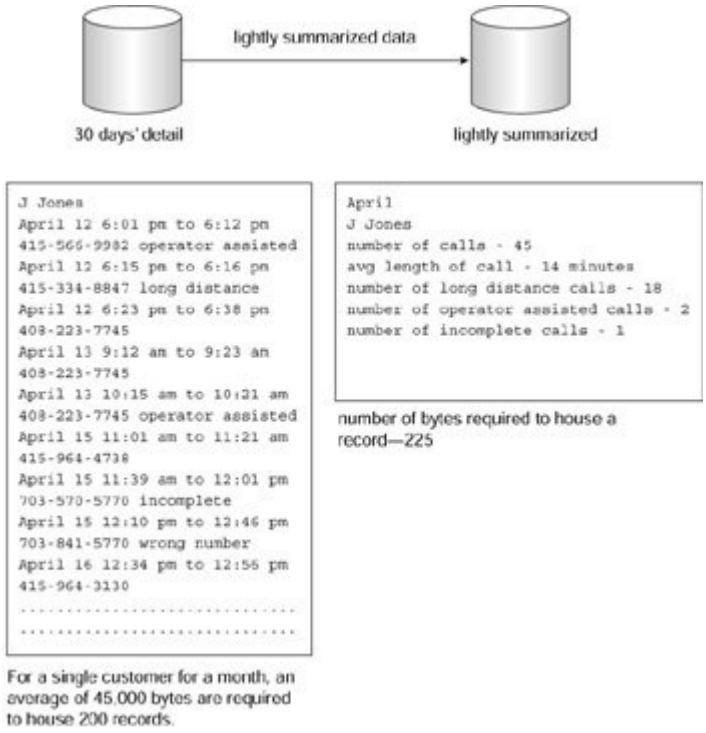

Figure 2.16: With light summarization data, large quantities of data can be represented compactly.

As data is passed from the operational, 30-day store of data, it is summarized, by customer, into fields that are likely to be used for DSS analysis. The record for J Jones shows the number of calls made per month, the average length of each call, the number of long-distance calls made, the number of operator-assisted calls, and so forth.

There is significantly less volume of data in the lightly summarized database than there is in the detailed database. Of course, there is a limit to the level of detail that can be accessed in the lightly summarized database.

The second tier of data in the data warehouse—the lowest level of granularity—is stored in the true archival level of data, as shown in Figure 2.17.

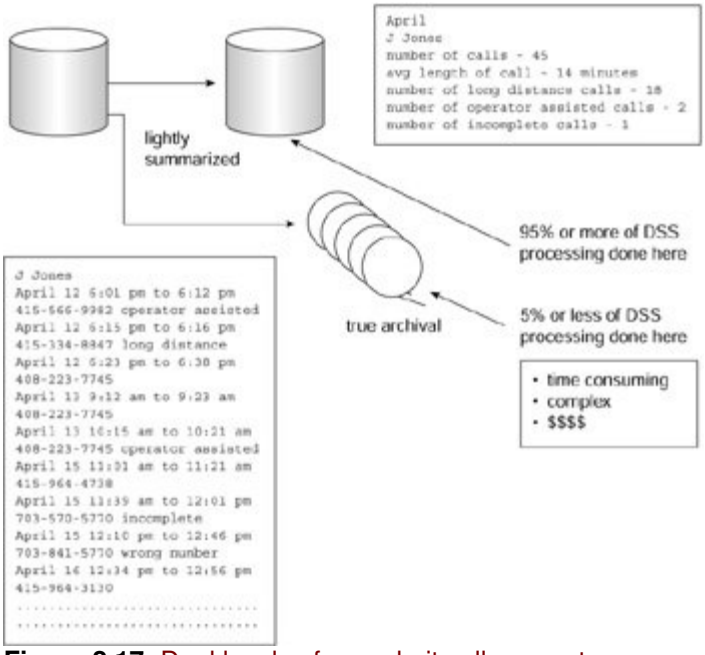

Figure 2.17: Dual levels of granularity allow you to process the majority of requests efficiently and answer any question that can be answered.

At the true archival level of data, all the detail coming from the operational environment is stored. There is truly a multitude of data at this level. For that reason, it makes sense to store the data on a medium such as magnetic tape or another bulk storage medium because the volume of data is so large.

By creating two levels of granularity in the data warehouse, the DSS architect has killed two birds with one stone—most DSS processing is performed against the lightly summarized data, where the data is compact and efficiently accessed. When some greater level of detail is required—5 percent of the time or less—there is the true archival level of data. It is expensive, cumbersome, and complex to access data at the true archival level of granularity, but if there it is there when necessary.

If a pattern of searching the true archival level of data develops over time, the designer may want to create some new fields of data at the lightly summarized level, so that most of the processing can occur there.

Because of the costs, efficiencies, ease of access, and ability to answer any query that can be answered, the dual level of data is the best architectural choice for the detailed level of the data warehouse for most shops. A single level of data should only be attempted when a shop has a relatively small amount of data in the data warehouse environment.

Exploration and Data Mining

The granular data found in the data warehouse supports more than data marts. It also supports the processes of exploration and data mining. Exploration and data mining take masses of detailed, historical data and examine it for previously unknown patterns of business activity.

The data warehouse contains a very useful source of data for the explorer and data miner. The data found in the data warehouse is cleansed, integrated, organized. And the data is historical. This foundation is precisely what the data miner and the explorer need in order to

start the exploration and data mining activity. It is noteworthy that while the data warehouse provides an excellent source of data for the miner and the explorer, the data warehouse is often not the only source. External data and other data can be freely mixed with data warehouse data in the course of doing exploration and mining. Refer to the book *Exploration Warehousing* (Wiley, 2000) for more information on this topic.

Living Sample Database

Occasionally, it is necessary to create a different kind of data warehouse. Sometimes there is simply too much data for normal access and analysis. When this happens, special design approaches may be used.

An interesting hybrid form of a data warehouse is the living sample database, which is useful when the volume of data in the warehouse has grown very large. The living sample database refers to a subset of either true archival data or lightly summarized data taken from a data warehouse. The term “living” stems from the fact that it is a subset—a sample—of a larger database, and the term “sample” stems from the fact that periodically the database needs to be refreshed. Figure 2.18 shows a living sample database.

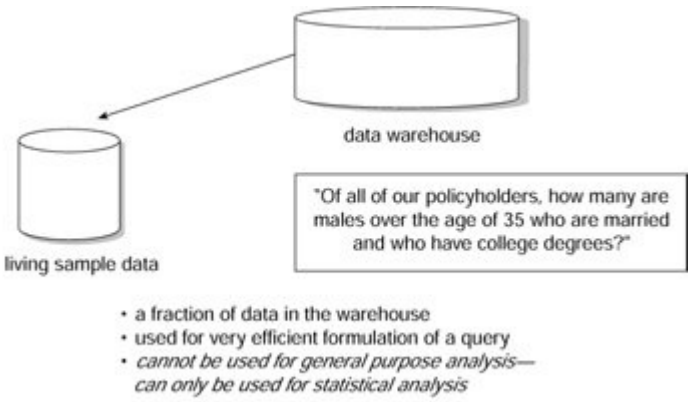

Figure 2.18: Living simple data-another way of changing the granularity of data.

In some circumstances (for example, statistical analysis of a population or profiling), a living sample database can be very useful and can save huge amounts of resources. But there are some severe restrictions, and the designer should not build such a database as part of the data warehouse unless he or she is aware of the limitations.

A living sample database is not a general-purpose database. If you wanted to find out whether J Jones is a customer, you would not look into a living sample database for that information. It is absolutely possible for J Jones to be a customer but not be on record in the living sample. These databases are good for statistical analysis and looking at trends, and can offer some very promising results when data must be looked at collectively. They are not at all useful for dealing with individual records of data.

One of the important aspects of the building a living sample database is how the data is loaded, which determines the amount of data in the database and how random the data will be. Consider how a living sample database is typically loaded. An extract/selection program rummages through a large database, choosing every one-hundredth or every one-thousandth record. The record is then shipped to the living sample database. The resulting living sample database, then, is one-hundredth or one-thousandth the size of the original database. The query that operates against this database then uses one-hundredth or one-

thousandth the resources as a query that would operate directly against the full data warehouse.

The selection of records for inclusion into the living sample is usually random. On occasion, a judgment sample is taken, in which a record must meet certain criteria in order to be selected. The problem with judgment samples is that they almost always introduce bias into the living sample data. The problem with a random selection of data is that it may not produce statistical significance. But however it's done, a subset of the data warehouse is selected for the living sample. The fact that any given record is not found in the living sample database means nothing because the processing that operates against the living sample does not require every record in the data warehouse to be in the living sample.

The greatest asset of a living sample database is that it is very efficient to access. Because its size is a fraction of the larger database from which it was derived, it is correspondingly much more efficient to access and analyze.

Put another way, suppose an analyst takes 24 hours to scan and analyze a large database. It may take as little as 10 minutes to scan and analyze a living sample database. In doing heuristic analysis, the turnaround time is crucial to the analysis that can be done. In heuristic analysis, the analyst runs a program, studies the results, reformulates the program, and runs it again. If it takes 24 hours to execute the program, the process of analysis and reformulation is greatly impaired (not to mention the resources required to do the reformulation).

With a living sample database small enough to be scanned in 10 minutes, the analyst can go through the iterative process very quickly. In short, the productivity of the DSS analyst depends on the speed of turning around the analysis being done.

One argument claims that doing statistical analysis yields incorrect answers. For example, an analyst may run against a large file of 25 million records to determine that 56.7 percent of the drivers on the road are men. Using a living sample database, the analyst uses 25,000 records to determine that 55.9 percent of the drivers on the road are men. One analysis has required vastly more resources than the other, yet the difference between the calculations is very small. Undoubtedly, the analysis against the large database was more accurate, but the cost of that accuracy is exorbitant, especially in the face of heuristic processing, where iterations of processing are the norm.

If very high degrees of accuracy are desired, a useful technique is to formulate the request and go through the iterative processing on the living sample database. In doing so, the DSS analyst quickly formulates the request. Then, after several iterations of analysis have been done, when the request is understood, it is run one final time against the large database.

Living sample data is just one more way of changing the level of granularity in the data warehouse to accommodate DSS processing.

Partitioning as a Design Approach

A second major design issue of data in the warehouse (after that of granularity) is that of partitioning (see Figure 2.11b). Partitioning of data refers to the breakup of data into separate physical units that can be handled independently. In the data warehouse, the issues surrounding partitioning do not focus on whether partitioning should be done but how it should be done.

It is often said that if both granularity and partitioning are done properly, then almost all other aspects of the data warehouse design and implementation come easily. If granularity is not

handled properly and if partitioning is not designed and implemented carefully, then no other aspects of design really matter.

Proper partitioning can benefit the data warehouse in several ways:

- Loading data
- Accessing data
- Archiving data
- Deleting data
- Monitoring data
- Storing data

Partitioning data properly allows data to grow and to be managed. Not partitioning data properly does not allow data to be managed or to grow gracefully.

Of course, there are other important design aspects of the data warehouse that will be discussed in later chapters.

Partitioning of Data

In the data warehouse environment, the question is not whether current detail data will be partitioned but how current detail data will be partitioned. Figure 2.19 illustrates partitioning.

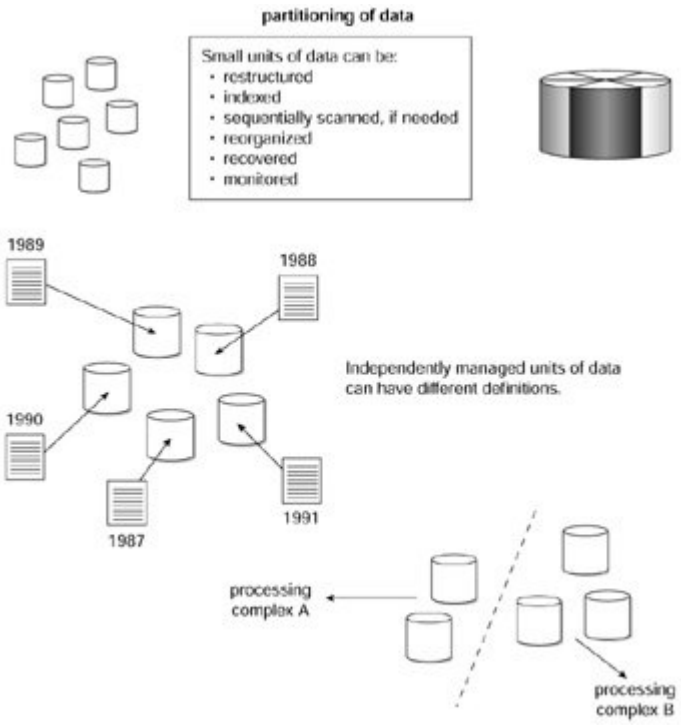

Figure 2.19: Independently managed partitions of data can be sent to different processing complexes with no other system considerations.

The purpose of partitioning of current detail data is to break data up into small, manageable physical units. Why is this so important? The operations staff and the designer have more flexibility in managing small physical units of data than large ones.

Following are some of the tasks that cannot easily be performed when data resides in large physical units:

- Restructuring
- Indexing
- Sequential scanning, if needed
- Reorganization
- Recovery
- Monitoring

In short, one of the essences of the data warehouse is the flexible access of data. Large masses of data defeat much of the purpose of the data warehouse. Therefore, all current-detail data warehouse data will be partitioned.

Data is partitioned when data of a like structure is divided into more than one physical unit of data. In addition, any given unit of data belongs to one and only one partition.

Data can be divided by many criteria, such as:

- By date
- By line of business
- By geography
- By organizational unit
- By all of the above

The choices for partitioning data are strictly up to the developer. In the data warehouse environment, however, it is almost mandatory that one of the criteria for partitioning be by date.

As an example of how a life insurance company may choose to partition its data, consider the following physical units of data:

- 2000 health claims
- 2001 health claims
- 2002 health claims
- 1999 life claims
- 2000 life claims
- 2001 life claims
- 2002 life claims
- 2000 casualty claims
- 2001 casualty claims
- 2002 casualty claims

The insurance company has used the criteria of date—that is, year—and type of claim to partition the data.

Partitioning can be done in many ways. One of the major issues facing the data warehouse developer is whether partition at the system level or at the application level. Partitioning at the system level is a function of the DBMS and the operating system to some extent. Partitioning at the application level is done by application code and is solely and strictly controlled by the developer and the programmer, so the DBMS and the system know of no relation between one partition and the other.

As a rule, it makes sense to partition data warehouse data at the application level. There are some important reasons for this. The most important is that at the application level there can be a different definition of data for each year. There can be 2000's definition of data and there can be 2001's definition of data, which may or may not be the same thing. The nature of data in a warehouse is the collection of data over a long period of time.

When the partitioning is done at the system level, the DBMS inevitably requires that there be a single definition of data. Given that the data warehouse holds data for a long period of time—up to 10 years—and given that the definition regularly changes, it does not make sense to allow the DBMS or the operating system to dictate that there should be a single definition of data.

By allowing the partitioning of data to be managed at the application level rather than the DBMS level, data can be moved from one processing complex to another with impunity. When the workload and volume of data become a real burden in the data warehouse environment, this feature may be a real advantage.

The acid test for the partitioning of data is to ask the question, “Can an index be added to a partition with no discernible interruption to other operations?” If an index can be added at will, then the partition is fine enough. If an index cannot be added easily, then the partition needs to be broken down more finely

Structuring Data in the Data Warehouse

So far, we haven’t gone into what the data structures found in the data warehouse really look like. Many kinds of structures are found in the data warehouse. We will look at some of the more common ones now.

Perhaps the simplest and most common data structure found in the data warehouse is the simple cumulative structure, shown in Figure 2.20.

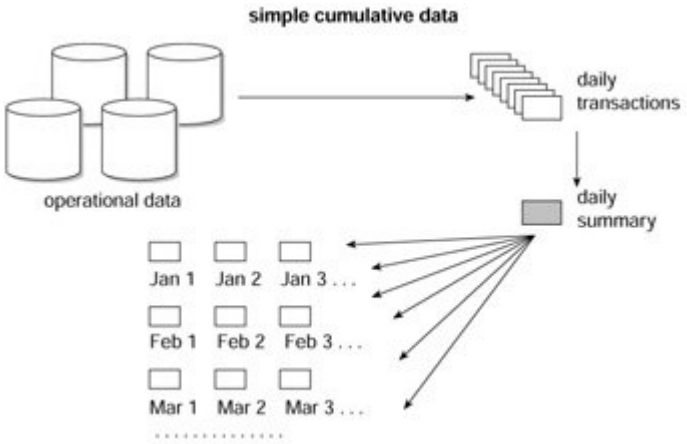

Figure 2.20: The simplest form of data in the data warehouse is the data that has been accumulated on a record-by-record basis, called simple cumulative data.

Figure 2.20 shows the daily transactions being transported from the operational environment. After that, they are summarized into data warehouse records, which may be by customer, by account, or by any subject area in the data warehouse. The transactions in Figure 2.20 are summarized by day. In other words, all daily activity for a customer for an account are totaled and passed into the data warehouse on a day-by-day basis.

Figure 2.21 shows a variation of the simple daily accumulation called the storage of rolling summary data.

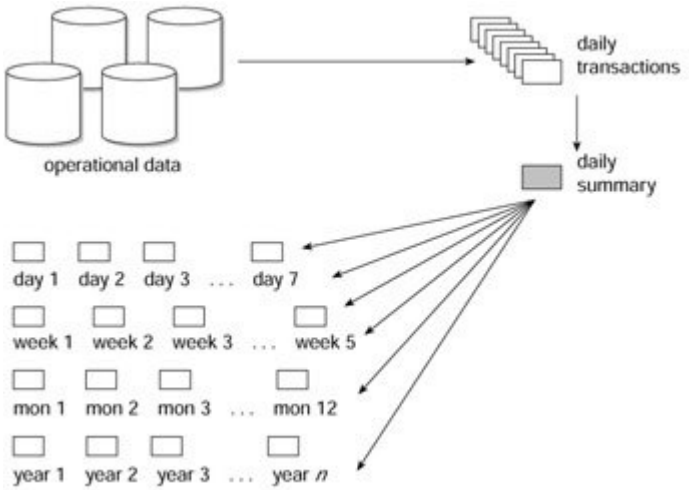

Figure 2.21: A variation of the cumulative file is the rolling summary file.

The data passes from the operational environment to the data warehouse environment as it did previously. In rolling summary data, however, the data is entered into a very different structure. For the first seven days of the week, activity is summarized into seven daily slots. On the eighth day, the seven daily slots are added together and placed into the first weekly slot. Then the daily totals are added into the first daily slot.

At the end of the month, the weekly slots are added together and placed in the first monthly slot. Then the weekly slots are reset to zero. At the end of the year, the monthly slots are added together, and the first yearly slot is loaded. Then the monthly slots are reset to zero.

A rolling summary data structure handles many fewer units of data than does a simple cumulative structuring of data. A comparison of the advantages and the disadvantages of rolling summary versus simple cumulative structuring of data is shown in Figure 2.22.

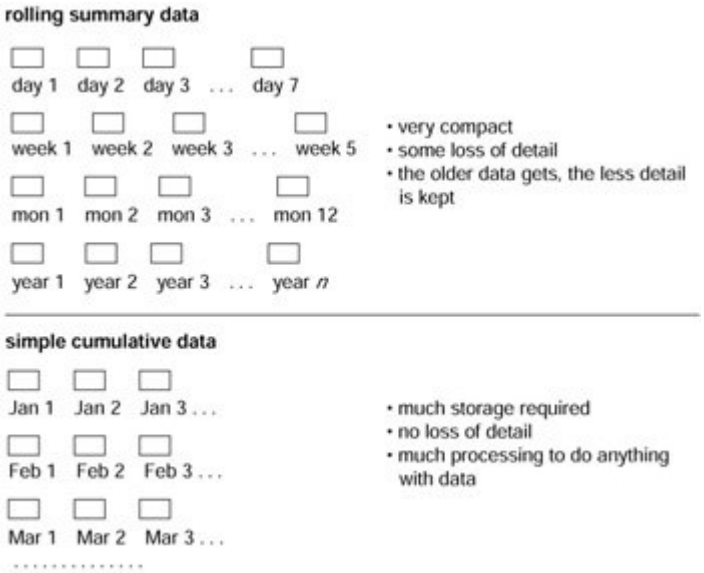

Figure 2-22: Comparing simple cumulative data with rolling summary data.

Another possibility for the structuring of data warehouse data is the simple direct file, shown in Figure 2.23.

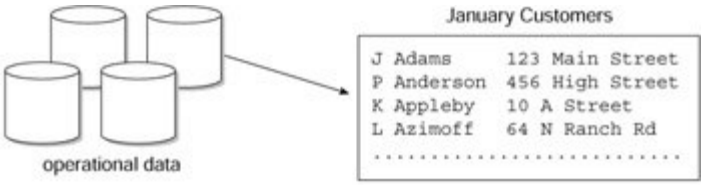

Figure 2.23: Creating a continuous file from direct files.

Figure 2.23 shows that data is merely pulled from the operational environment to the data warehouse environment; there is no accumulation. In addition, the simple direct file is not done on a daily basis. Instead, it is done over a longer period of time, such as a week or a month. As such, the simple direct file represents a snapshot of operational data taken as of one instant in time.

A continuous file can be created from two or more simple direct files, as shown in Figure 2.24. Two snapshots—one from January and one from February—are merged to create a continuous file of data. The data in the continuous file represents the data continuously from the first month to the last.

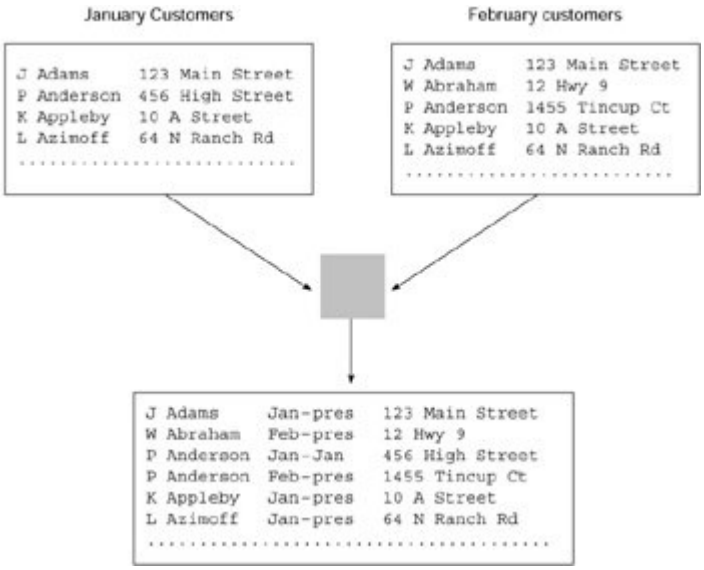

Figure 2.24: Creating a continuous file from direct files.

Of course, a continuous file can be created by appending a snapshot of data onto a previous version of the continuous file, as shown in Figure 2.25.

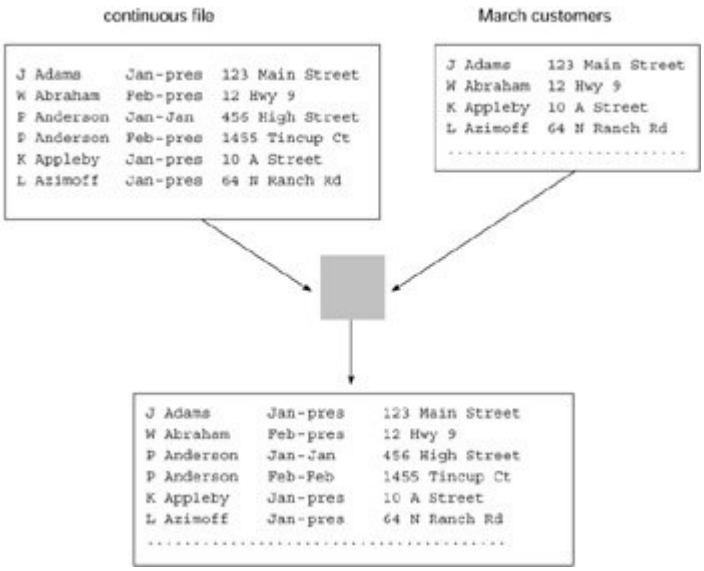

Figure 2.25: Continuous files can be created from simple direct files, or they may have simple direct files appended to them.

There are many more ways to structure data within the data warehouse. The most common are these:

- Simple cumulative
- Rolling summary
- Simple direct
- Continuous

At the key level, data warehouse keys are inevitably compounded keys. There are two compelling reasons for this:

- Date—year, year/month, year/month/day, and so on—is almost always a part of the key.
- Because data warehouse data is partitioned, the different components of the partitioning show up as part of the key.

Data Warehouse: The Standards Manual

The data warehouse is relevant to many people-managers, DSS analysts, developers, planners, and so forth. In most organizations, the data warehouse is new. Accordingly, there should be an official organizational explanation and description of what is in the data warehouse and how the data warehouse can be used.

Calling the explanation of what is inside the data warehouse a “standards manual” is probably deadly. Standards manuals have a dreary connotation and are famous for being ignored and gathering dust. Yet, some form of internal publication is a necessary and worthwhile endeavor.

The kinds of things the publication (whatever it is called!) should contain are the following:

- A description of what a data warehouse is
- A description of source systems feeding the warehouse
- How to use the data warehouse
- How to get help if there is a problem
- Who is responsible for what
- The migration plan for the warehouse
- How warehouse data relates to operational data

- How to use warehouse data for DSS
- When not to add data to the warehouse
- What kind of data is not in the warehouse
- A guide to the meta data that is available
- What the system of record is

Auditing and the Data Warehouse

An interesting issue that arises with data warehouses is whether auditing can be or should be done from them. Auditing can be done from the data warehouse. In the past there have been a few examples of detailed audits being performed there. But there are many reasons why auditing—even if it can be done from the data warehouse—should not be done from there. The primary reasons for not doing so are the following:

- Data that otherwise would not find its way into the warehouse suddenly has to be there.
- The timing of data entry into the warehouse changes dramatically when auditing capability is required.
- The backup and recovery restrictions for the data warehouse change drastically when auditing capability is required.
- Auditing data at the warehouse forces the granularity of data in the warehouse to be at the very lowest level.

In short, it is possible to audit from the data warehouse environment, but due to the complications involved, it makes much more sense to audit elsewhere.

Cost Justification

Cost justification for the data warehouse is normally not done on an a priori, return-on-investment (ROI) basis. To do such an analysis, the benefits must be known prior to building the data warehouse.

In most cases, the real benefits of the data warehouse are not known or even anticipated before construction begins because the warehouse is used differently than other data and systems built by information systems. Unlike most information processing, the data warehouse exists in a realm of “Give me what I say I want, then I can tell you what I really want.” The DSS analyst really cannot determine the possibilities and potentials of the data warehouse, nor how and why it will be used, until the first iteration of the data warehouse is available. The analyst operates in a mode of discovery, which cannot commence until the data warehouse is running in its first iteration. Only then can the DSS analyst start to unlock the potential of DSS processing.

For this reason, classical ROI techniques simply do not apply to the data warehouse environment. Fortunately, data warehouses are built incrementally. The first iteration can be done quickly and for a relatively small amount of money. Once the first portion of the data warehouse is built and populated, the analyst can start to explore the possibilities. It is at this point that the analyst can start to justify the development costs of the warehouse.

As a rule of thumb, the first iteration of the data warehouse should be small enough to be built and large enough to be meaningful. Therefore, the data warehouse is best built a small iteration at a time. There should be a direct feedback loop between the warehouse developer and the DSS analyst, in which they are constantly modifying the existing warehouse data and adding other data to the warehouse. And the first iteration should be done quickly. It is said that the initial data warehouse design is a success if it is 50 percent accurate.

Typically, the initial data warehouse focuses on one of these functional areas:

- Finance
- Marketing
- Sales

Occasionally, the data warehouse's first functional area will focus on one of these areas:

- Engineering/manufacturing
- Actuarial interests

Justifying Your Data Warehouse

There is no getting around the fact that data warehouses cost money. Data, processors, communications, software, tools, and so forth all cost money. In fact, the volumes of data that aggregate and collect in the data warehouse go well beyond anything the corporation has ever seen. The level of detail and the history of that detail all add up to a large amount of money.

In almost every other aspect of information technology, the major investment for a system lies in creating, installing, and establishing the system. The ongoing maintenance costs for a system are miniscule compared to the initial costs. However, establishing the initial infrastructure of the data warehouse is not the most significant cost—the ongoing maintenance costs far outweigh the initial infrastructure costs. There are several good reasons why the costs of a data warehouse are significantly different from the cost of a standard system:

- The truly enormous volume of data that enters the data warehouse.
- The cost of maintaining the interface between the data warehouse and the operational sources. If the organization has chosen an extract/transfer/load (ETL) tool, then these costs are mitigated over time; if an organization has chosen to build the interface manually, then the costs of maintenance skyrocket.
- The fact that a data warehouse is never done. Even after the initial few iterations of the data warehouse are successfully completed, adding more subject areas to the data warehouse is an ongoing need.

Cost of Running Reports

How does an organization justify the costs of a data warehouse before the data warehouse is built? There are many approaches. We will discuss one in depth here, but be advised that there are many other ways to justify a data warehouse.

We chose this approach because it is simple and because it applies to every organization. When the justification is presented properly, it is very difficult to deny the powerful cost justifications for a data warehouse. It is an argument that technicians and non-technicians alike can appreciate and understand.

Data warehousing lowers the cost of information by approximately two orders of magnitude. This means that with a data warehouse an organization can access a piece of information for $100; an organization that does not have a data warehouse can access the same unit of information for $10,000.

How do you show that data warehousing greatly lowers the cost of information? First, use a report. This doesn't necessarily need to be an actual report. It can be a screen, a report, a spreadsheet, or some form of analytics that demonstrates the need for information in the corporation. Second, you should look at your legacy environment, which includes single or multiple applications, old and new applications. The applications may be Enterprise

Resource Planning (ERP) applications, non-ERP applications, online applications, or offline applications.

Now consider two companies, company A and company B. The companies are identical in respect to their legacy applications and their need for information. The only difference between the two is that company B has a data warehouse from which to do reporting and company A does not.

Company A looks to its legacy applications to gather information. This task includes the following:

- Finding the data needed for the report
- Accessing the data
- Integrating the data
- Merging the data
- Building the report

Finding the data can be no small task. In many cases, the legacy systems are not documented. There is a time-honored saying: Real programmers don't do documentation. This will come back to haunt organizations, as there simply is no easy way to go back and find out what data is in the old legacy systems and what processing has occurred there.

Accessing the data is even more difficult. Some of the legacy data is in Information Management System (IMS), some in Model 204, some in Adabas. And there is no IMS, Model 204, and Adabas technology expertise around anymore. The technology that houses the legacy environment is a mystery. And even if the legacy environment can be accessed, the computer operations department stands in the way because it does not want anything in the way of the online window of processing.

If the data can be found and accessed, it then needs to be integrated. Reports typically need information from multiple sources. The problem is that those sources were never designed to be run together. A customer in one system is not a customer in another system, a transaction in one system is different from a transaction in another system, and so forth. A tremendous amount of conversion, reformatting, interpretation, and the like must go on in order to integrate data from multiple systems.

Merging the data is easy in some cases. But in the case of large amounts of data or in the case of data coming from multiple sources, the merger of data can be quite an operation.

Finally, the report is built.

How long does this process take for company A? How much does it cost? Depending on the information that is needed and depending on the size and state of the legacy systems environment, it may take a considerable amount of time and a high cost to get the information. The typical cost ranges from $25,000 to $1 million. The typical length of time to access data is anywhere from 1 to 12 months.

Now suppose that an company B has built a data warehouse. The typical cost here ranges from $500 to $10,000. The typical length of time to access data is one hour to a half day. We see that company B's costs and time investment for retrieving information are much lower. The cost differential between company A and company B forms the basis of the cost justification for a data warehouse. Data warehousing greatly lowers the cost of information and accelerates the time required to get the information.

Cost of Building the Data Warehouse

The astute observer will ask, what about the cost of building the data warehouse? Figure 2.26 shows that in order to generate a single report for company B, it is still necessary to find, access, integrate, and merge the data. These are the same initial steps taken to build a single report for company A, so there are no real savings found in building a data warehouse. Actually, building a data warehouse to run one report is a costly waste of time and money.

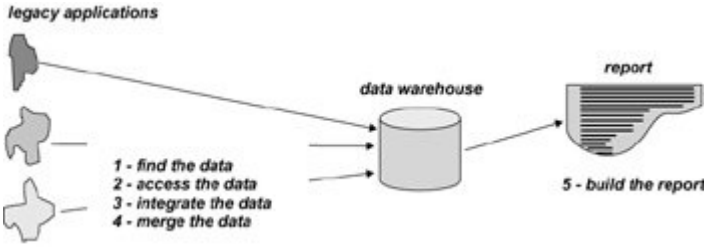

Figure 2.26: Where the costs and the activities are when a data warehouse is built.

But no corporation in the world operates from a single report. Different divisions of even the simplest, smallest corporation look at data differently. Accounting looks at data one way; marketing looks at data another way; sales looks at data yet another way; and management looks at data in even another way. In this scenario, the cost of building the data warehouse *is* worthwhile. It is a one-time cost that liberates the information found in the data warehouse. Whereas *each* report company A needs is both costly and time-consuming, company B uses the one-time cost of building the data warehouse to generate multiple reports (see Figure 2.27).

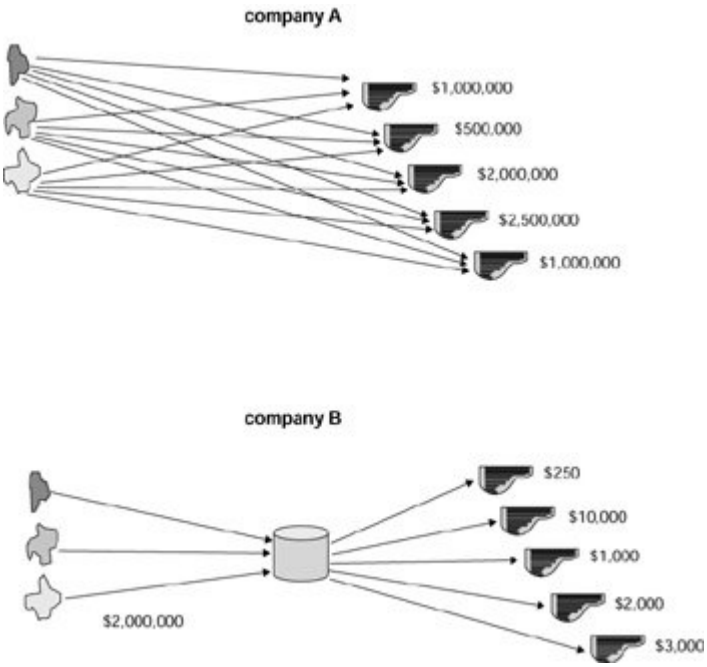

Figure 2.27: Multiple reports make the cost of the data warehouse worthwhile.

But that expense is a one-time expense, for the most part. (At least the initial establishment of the data warehouse is a one-time expense.) Figure 2.27 shows that indeed data warehousing greatly lowers the cost of information and greatly accelerates the rate at which information can be retrieved.

Would company A actually even pay to generate individual reports? Probably not. Perhaps it would pay the price for information the first few times. When it realizes that it cannot afford to pay the price for every report, it simply stops creating reports. The end user has the attitude, "I know the information is in my corporation, but I just can't get to it." The result of the high costs of getting information and the length of time required is such that end users are frustrated and are unhappy with their IT organization for not being able to deliver information.

Data Homogeneity/Heterogeneity

At first glance, it may appear that the data found in the data warehouse is homogeneous in the sense that all of the types of records are the same. In truth, data in the data warehouse is very heterogeneous. The data found in the data warehouse is divided into major subdivisions called subject areas. Figure 2.28 shows that a data warehouse has subject areas of product, customer, vendor, and transaction.

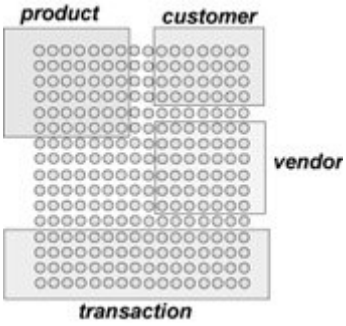

Figure 2.28: The data in the different parts of the data warehouse are grouped by subject area.

The first division of data inside a data warehouse is along the lines of the major subjects of the corporation. But with each subject area there are further subdivisions. Data within a subject area is divided into tables. Figure 2.29 shows this division of data into tables for the subject area product.

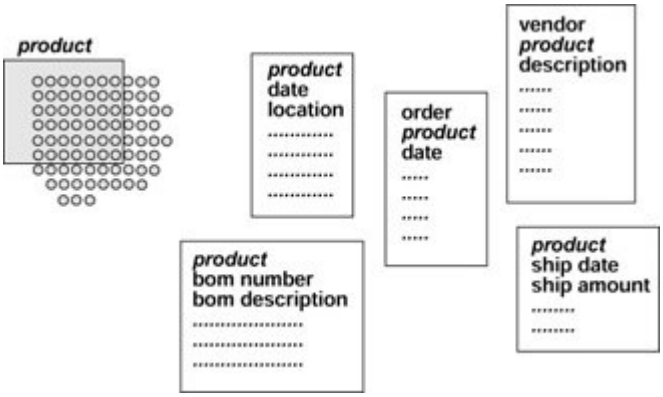

Figure 2.29: Within the product subject area there are different types of tables, but each table has a common product identifier as part of the key.

Figure 2.29 shows that there are five tables that make up the subject area inside the data warehouse. Each of the tables has its own data, and there is a common thread for each of

the tables in the subject area. That common thread is the key/foreign key data element—product.

Within the physical tables that make up a subject area there are further subdivisions. These subdivisions are created by different occurrences of data values. For example, inside the product shipping table, there are January shipments, February shipments, March shipments, and so forth.

The data in the data warehouse then is subdivided by the following criteria:

- Subject area
- Table
- Occurrences of data within table

This organization of data within a data warehouse makes the data easily accessible and understandable for all the different components of the architecture that must build on the data found there. The result is that the data warehouse, with its granular data, serves as a basis for many different components, as seen in Figure 2.30.

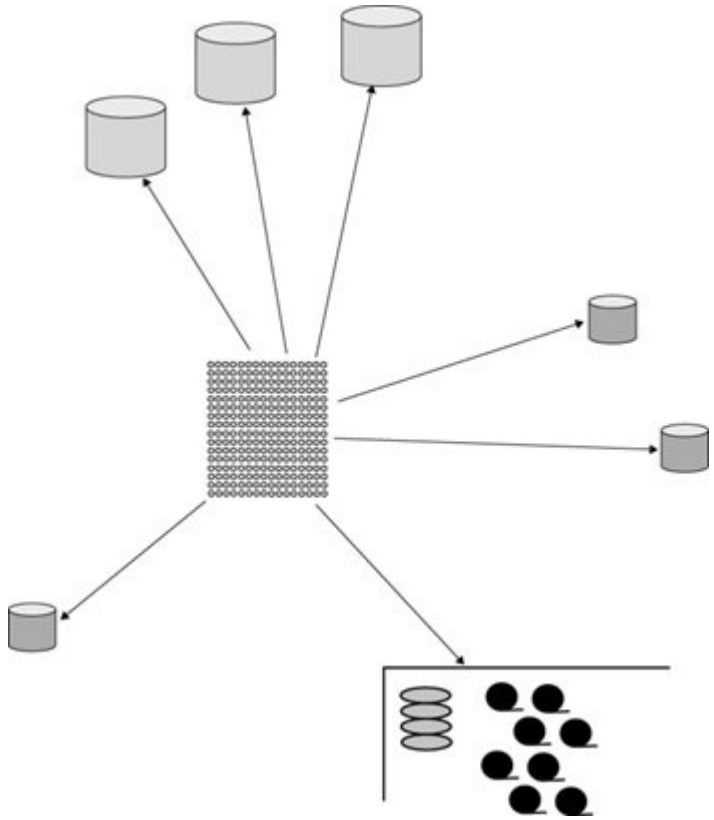

Figure 2.30: The data warehouse sits at the center of a large framework.

The simple yet elegant organization of data within the data warehouse environment seen in Figure 2.30 makes data accessible in many different ways for many different purposes.

Purging Warehouse Data

Data does not just eternally pour into a data warehouse. It has its own life cycle within the warehouse as well. At some point in time, data is purged from the warehouse. The issue of

purging data is one of the fundamental design issues that must not escape the data warehouse designer.

In some senses, data is not purged from the warehouse at all. It is simply rolled up to higher levels of summary. There are several ways in which data is purged or the detail of data is transformed, including the following:

- Data is added to a rolling summary file where detail is lost.
- Data is transferred to a bulk storage medium from a high-performance medium such as DASD.
- Data is actually purged from the system.
- Data is transferred from one level of the architecture to another, such as from the operational level to the data warehouse level.

There are, then, a variety of ways in which data is purged or otherwise transformed inside the data warehouse environment. The life cycle of data—including its purge or final archival dissemination—should be an active part of the design process for the data warehouse.

Reporting and the Architected Environment

It is a temptation to say that once the data warehouse has been constructed all reporting and informational processing will be done from there. That is simply not the case. There is a legitimate class of report processing that rightfully belongs in the domain of operational systems. Figure 2.31 shows where the different styles of processing should be located.

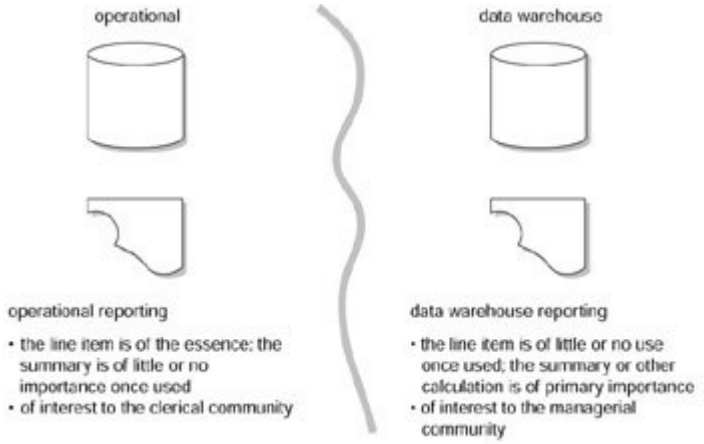

Figure 2.31: The differences between the two types of reporting.

Figure 2.31 shows that operational reporting is for the clerical level and focuses primarily on the line item. Data warehouse or informational processing focuses on management and contains summary or otherwise calculated information. In the data warehouse style of reporting, little use is made of line-item, detailed information, once the basic calculation of data is made.

As an example of the differences between operational reporting and DSS reporting, consider a bank. Every day before going home a teller must balance the cash in his or her window. This means that the teller takes the starting amount of cash, tallies all the day's transactions, and determines what the day's ending cash balance should be. In order to do this, the teller needs a report of all the day's transactions. This is a form of operational reporting.

Now consider the bank vice president who is trying to determine how many new ATMs to place in a newly developed shopping center. The banking vice president looks at a whole host of information, some of which comes from within the bank and some of which comes from outside the bank. The bank vice president is making a long-term, strategic decision and uses classical DSS information for his or her decision.

There is then a real difference between operational reporting and DSS reporting. Operational reporting should always be done within the confines of the operational environment.

The Operational Window of Opportunity

In its broadest sense, archival represents anything older than right now. Thus, the loaf of bread that I bought 30 seconds ago is archival information. The only thing that is not archival is information that is current.

The foundation of DSS processing—the data warehouse—contains nothing but archival information, most of it at least 24 hours old. But archival data is found elsewhere throughout the architected environment. In particular, some limited amounts of archival data are also found in the operational environment.

In the data warehouse it is normal to have a vast amount of archival data—from 5 to 10 years of data is common. Because of the wide time horizon of archival data, the data warehouse contains a massive amount of data. The time horizon of archival data found in the operational environment—the "operational window" of data—is not nearly as long. It can be anywhere from 1 week to 2 years.

The time horizon of archival data in the operational environment is not the only difference between archival data in the data warehouse and in the operational environment. Unlike the data warehouse, the operational environment's archival data is nonvoluminous and has a high probability of access.

In order to understand the role of fresh, nonvoluminous, high-probability-of-access archival data in the operational environment, consider the way a bank works. In a bank environment, the customer can reasonably expect to find information about this month's transactions. Did this month's rent check clear? When was a paycheck deposited? What was the low balance for the month? Did the bank take out money for the electricity bill last week?

The operational environment of a bank, then, contains very detailed, very current transactions (which are still archival). Is it reasonable to expect the bank to tell the customer whether a check was made out to the grocery store 5 years ago or whether a check to a political campaign was cashed 10 years ago? These transactions would hardly be in the domain of the operational systems of the bank. These transactions very old, and so the has a very low probability of access.

The operational window of time varies from industry to industry and even in type of data and activity within an industry.

For example, an insurance company would have a very lengthy operational window—from 2 to 3 years. The rate of transactions in an insurance company is very low, at least compared to other types of industries. There are relatively few direct interactions between the customer and the insurance company. The operational window for the activities of a bank, on the other hand, is very short—from 0 to 60 days. A bank has many direct interactions with its customers.

The operational window of a company depends on what industry the company is in. In the case of a large company, there may be more than one operational window, depending on the particulars of the business being conducted. For example, in a telephone company, customer usage data may have an operational window of 30 to 60 days, while vendor/supplier activity may have a window of 2 to 3 years.

The following are some suggestions as to how the operational window of archival data may look in different industries:

- Insurance—2 to 3 years
- Bank trust processing—2 to 5 years
- Telephone customer usage—30 to 60 days
- Supplier/vendor activity—2 to 3 years
- Retail banking customer account activity—30 days
- Vendor activity—1 year
- Loans—2 to 5 years
- Retailing SKU activity—1 to 14 days
- Vendor activity—1 week to 1 month
- Airlines flight seat activity— 30 to 90 days
- Vendor/supplier activity—1 to 2 years
- Public utility customer utilization—60 to 90 days
- Supplier activity—1 to 5 years

The length of the operational window is very important to the DSS analyst because it determines where the analyst goes to do different kinds of analysis and what kinds of analysis can be done. For example, the DSS analyst can do individual-item analysis on data found within the operational window, but cannot do massive trend analysis over a lengthy period of time. Data within the operational window is geared to efficient individual access. Only when the data passes out of the operational window is it geared to mass data storage and access.

On the other hand, the DSS analyst can do sweeping trend analysis on data found outside the operational window. Data out there can be accessed and processed en masse, whereas access to any one individual unit of data is not optimal.

Incorrect Data in the Data Warehouse

The architect needs to know what to do about incorrect data in the data warehouse. The first assumption is that incorrect data arrives in the data warehouse on an exception basis. If data is being incorrectly entered in the data warehouse on a wholesale basis, then it is incumbent on the architect to find the offending ETL program and make adjustments. Occasionally, even with the best of ETL processing, a few pieces of incorrect data enter the data warehouse environment. How should the architect handle incorrect data in the data warehouse?

There are at least three options. Each approach has its own strengths and weaknesses, and none are absolutely right or wrong. Instead, under some circumstances one choice is better than another.

For example, suppose that on July 1 an entry for $5,000 is made into an operational system for account ABC. On July 2 a snapshot for $5,000 is created in the data warehouse for account ABC. Then on August 15 an error is discovered. Instead of an entry for $5,000, the entry should have been for $750. How can the data in the data warehouse be corrected?

- Choice 1: Go back into the data warehouse for July 2 and find the offending entry. Then, using update capabilities, replace the value $5,000 with the value $750. This is a clean and neat solution when it works, but it introduces new issues:

        - The integrity of the data has been destroyed. Any report running between July 2 and Aug 16 will not be able to be reconciled.
        - The update must be done in the data warehouse environment.
        - In many cases there is not a single entry that must be corrected, but many, many entries that must be corrected.
    - Choice 2: Enter offsetting entries. Two entries are made on August 16, one for 2$5,000 and another for 1$750. This is the best reflection of the most up-to-date information in the data warehouse between July 2 and August 16. There are some drawbacks to this approach:
        - Many entries may have to be corrected, not just one. Making a simple adjustment may not be an easy thing to do at all.
        - Sometimes the formula for correction is so complex that making an adjustment cannot be done.
    - Choice 3: Reset the account to the proper value on August 16. An entry on August 16 reflects the balance of the account at that moment regardless of any past activity. An entry would be made for $750 on August 16. But this approach has its own drawbacks:
        - The ability to simply reset an account as of one moment in time requires application and procedural conventions.
        - Such a resetting of values does not accurately account for the error that has been made.

Choice 3 is what likely happens when you cannot balance your checking account at the end of the month. Instead of trying to find out what the bank has done, you simply take the bank's word for it and reset the account balance.

There are then at least three ways to handle incorrect data as it enters the data warehouse. Depending on the circumstances, one of the approaches will yield better results than another approach.

Summary

The two most important design decisions that can be made concern the granularity of data and the partitioning of data. For most organizations, a dual level of granularity makes the most sense. Partitioning of data breaks it down into small physical units. As a rule, partitioning is done at the application level rather than at the system level.

Data warehouse development is best done iteratively. First one part of the data warehouse is constructed, then another part of the warehouse is constructed. It is *never* appropriate to develop the data warehouse under the "big bang" approach. One reason is that the end user of the warehouse operates in a discovery mode, so only *after* the warehouse's first iteration is built can the developer tell what is really needed in the warehouse.

The granularity of the data residing inside the data warehouse is of the utmost importance. A very low level of granularity creates too much data, and the system is overwhelmed by the volumes of data. A very high level of granularity is efficient to process but precludes many kinds of analyses that need detail. In addition, the granularity of the data warehouse needs to be chosen in an awareness of the different architectural components that will feed off the data warehouse.

Surprisingly, many design alternatives can be used to handle the issue of granularity. One approach is to build a multitiered data warehouse with dual levels of granularity that serve different types of queries and analysis. Another approach is to create a living sample database where statistical processing can be done very efficiently from a living sample database.

Partitioning a data warehouse is very important for a variety of reasons. When data is partitioned it can be managed in separate, small, discrete units. This means that loading the data into the data warehouse will be simplified, building indexes will be streamlined, archiving data will be easy, and so forth. There are at least two ways to partition data—at the DBMS/operating system level and at the application level. Each approach to partitioning has its own set of advantages and disadvantages.

Each unit of data in the data warehouse environment has a moment associated with it. In some cases, the moment in time appears as a snapshot on every record. In other cases, the moment in time is applied to an entire table. Data is often summarized by day, month, or quarter. In addition, data is created in a continuous manner. The internal time structuring of data is accomplished in many ways.

Auditing can be done from a data warehouse, but auditing should not be done from a data warehouse. Instead, auditing is best done in the detailed operational transaction-oriented environment. When auditing is done in the data warehouse, data that would not otherwise be included is found there, the timing of the update into the data warehouse becomes an issue, and the level of granularity in the data warehouse is mandated by the need for auditing, which may not be the level of granularity needed for other processing.

A normal part of the data warehouse life cycle is that of purging data. Often, developers neglect to include purging as a part of the specification of design. The result is a warehouse that grows eternally, which, of course, is an impossibility.

Chapter 3: The Data Warehouse and Design

Overview

There are two major components to building a data warehouse: the design of the interface from operational systems and the design of the data warehouse itself. Yet, "*design*" is not entirely accurate because it suggests planning elements out in advance. The requirements for the data warehouse cannot be known until it is partially populated and in use and design approaches that have worked in the past will not necessarily suffice in subsequent data warehouses. Data warehouses are constructed in a heuristic manner, where one phase of development depends entirely on the results attained in the previous phase. First, one portion of data is populated. It is then used and scrutinized by the DSS analyst. Next, based on feedback from the end user, the data is modified and/or other data is added. Then another portion of the data warehouse is built, and so forth.This feedback loop continues throughout the entire life of the data warehouse.

Therefore, data warehouses cannot be designed the same way as the classical requirements-driven system. On the other hand, anticipating requirements is still important. Reality lies somewhere in between.

Note: A data warehouse design methodology that parallels this chapter can be found—for free—on *www.billinmon.com*. The methodology is iterative and all of the required design steps are greatly detailed.

Beginning with Operational Data

At the outset, operational transaction-oriented data is locked up in existing legacy systems. Though tempting to think that creating the data warehouse involves only extracting operational data and entering it into the warehouse, nothing could be further from the truth. Merely pulling data out of the legacy environment and placing it in the data warehouse achieves very little of the potential of data warehousing.

Figure 3.1 shows a simplification of how data is transferred from the existing legacy systems environment to the data warehouse. We see here that multiple applications contribute to the data warehouse.

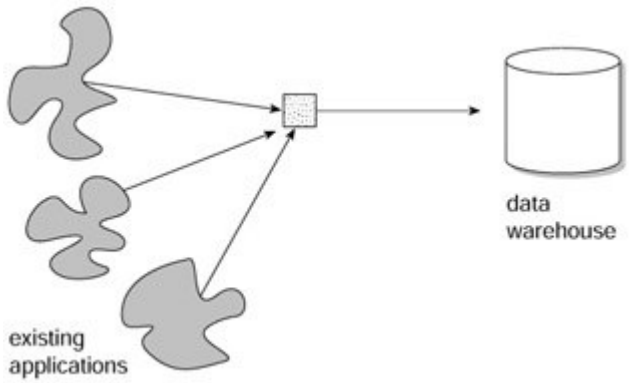

Figure 3.1: Moving from the operational to the data warehouse environment is not as simple as mere extraction.

Figure 3.1 is overly simplistic for many reasons. Most importantly, it does not take into account that the data in the operational environment is unintegrated. Figure 3.2 shows the

lack of integration in a typical existing systems environment. Pulling the data into the data warehouse without integrating it is a grave mistake.

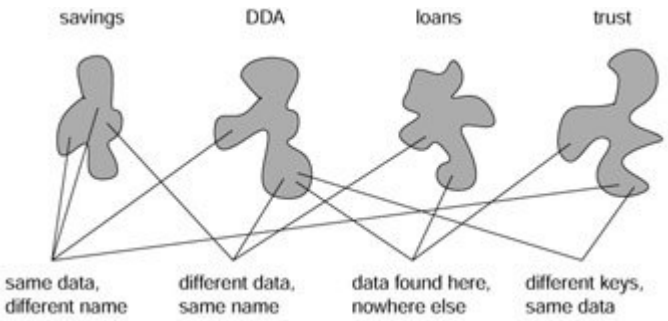

Figure 3.2: Data across the different applications is severely unintegrated.

When the existing applications were constructed, no thought was given to possible future integration. Each application had its own set of unique and private requirements. It is no surprise, then, that some of the same data exists in various places with different names, some data is labeled the same way in different places, some data is all in the same place with the same name but reflects a different measurement, and so on. Extracting data from many places and integrating it into a unified picture is a complex problem.

This lack of integration is the extract programmer's nightmare. As illustrated in Figure 3.3, countless details must be programmed and reconciled just to bring the data properly from the operational environment.

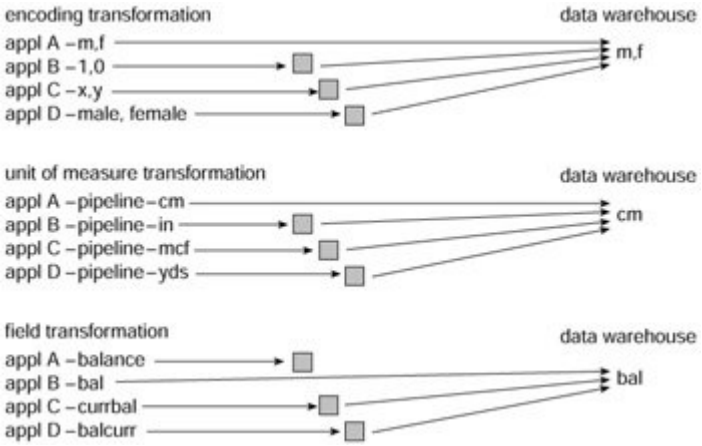

Figure 3.3: To properly move data from the existing systems environment to the data warehouse environment, it must be integrated.

One simple example of lack of integration is data that is not encoded consistently, as shown by the encoding of gender. In one application, gender is encoded as m/f. In another, it is encoded as 0/1. In yet another it is encoded as x/y. Of course, it doesn't matter how gender is encoded as long as it is done consistently. As data passes to the data warehouse, the applications' different values must be correctly deciphered and recoded with the proper value.

As another example, consider four applications that have the same field-pipeline. The pipeline field is measured differently in each application. In one application, pipeline is measured in inches, in another in centimeters, and so forth. It does not matter how pipeline

is measured in the data warehouse, as long as it is measured consistently. As each application passes its data to the warehouse, the measurement of pipeline is converted into a single consistent corporate measurement.

Field transformation is another integration issue. Say that the same field exists in four applications under four different names. To transform the data to the data warehouse properly, a mapping from the different source fields to the data warehouse fields must occur.

Yet another issue is that legacy data exists in many different formats under many different DBMSs. Some legacy data is under IMS, some legacy data is under DB2, and still other legacy data is under VSAM. But all of these technologies must have the data they protect brought forward into a single technology. Such a translation of technology is not always straightforward.

These simple examples hardly scratch the surface of integration, and they are not complex in themselves. But when they are multiplied by the thousands of existing systems and files, compounded by the fact that documentation is usually out-of-date or nonexistent, the issue of integration becomes burdensome.

But integration of existing legacy systems is not the only difficulty in the transformation of data from the operational, existing systems environment to the data warehouse environment. Another major problem is the efficiency of accessing existing systems data. How does the program that scans existing systems know whether a file has been scanned previously? The existing systems environment holds tons of data, and attempting to scan all of it every time a data warehouse load needs to be done is wasteful and unrealistic.

Three types of loads are made into the data warehouse from the operational environment:

- Archival data
- Data currently contained in the operational environment
- Ongoing changes to the data warehouse environment from the changes (updates) that have occurred in the operational environment since the last refresh

As a rule, loading archival data from the legacy environment as the data warehouse is first loaded presents a minimal challenge for two reasons. First, it often is not done at all. Organizations find the use of old data not cost-effective in many environments. Second, even when archival data is loaded, it is a one-time-only event.

Loading current, nonarchival data from the existing operational environment likewise presents a minimal challenge because it needs to be done only once. Usually, the existing systems environment can be downloaded to a sequential file, and the sequential file can be downloaded into the warehouse with no disruption to the online environment. Although system resources are required, because the process is done only once, the event is minimally disruptive.

Loading data on an ongoing basis—as changes are made to the operational environment—presents the largest challenge to the data architect. Efficiently trapping those ongoing daily changes and manipulating them is not easy. Scanning existing files, then, is a major issue facing the data warehouse architect.

Five common techniques are used to limit the amount of operational data scanned at the point of refreshing the data warehouse, as shown in Figure 3.4. The first technique is to scan data that has been timestamped in the operational environment. When an application stamps the time of the last change or update on a record, the data warehouse scan can run quite efficiently because data with a date other than that applicable does not have to be touched. It usually is only by happenstance, though, that existing data has been timestamped.

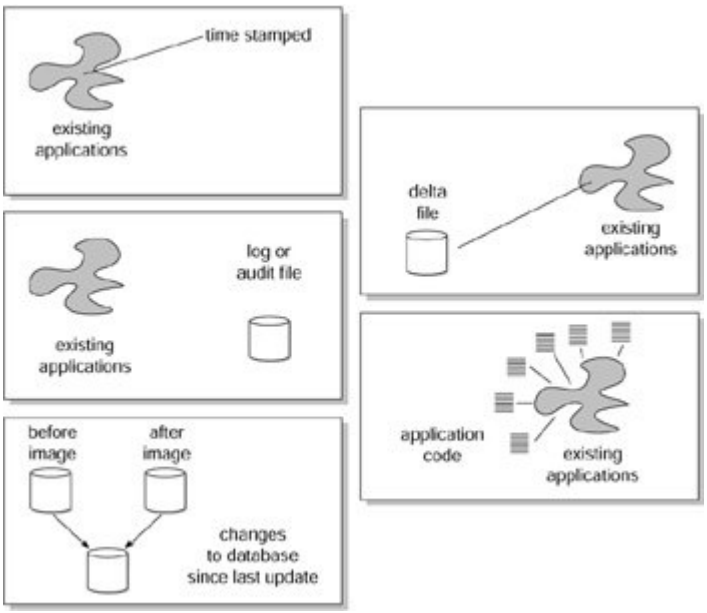

Figure 3.4: How do you know what source data to scan? Do you scan every record every day? Every week?

The second technique to limiting the data to be scanned is to scan a “delta” file. A delta file contains only the changes made to an application as a result of the transactions that have run through the operational environment. With a delta file, the scan process is very efficient because data that is not a candidate for scanning is never touched. Not many applications, however, build delta files.

The third technique is to scan a log file or an audit file created as a by-product of transaction processing. A log file contains essentially the same data as a delta file; however, there are some major differences. Many times, operations protects the log files because they are needed in the recovery process. Computer operations is not particularly thrilled to have its log file used for something other than its primary purpose. Another difficulty with a log tape is that the internal format is built for systems purposes, not applications purposes. A technological guru may be needed to interface the contents of data on the log tape. Another shortcoming is that the log file usually contains much more information than that desired by the data warehouse developer. Audit files have many of the same shortcomings as log files. An example of the use of log files to update a data warehouse is the Web logs created by the Web-based ebusiness environment.

The fourth technique for managing the amount of data scanned is to modify application code. This is never a popular option, as much application code is old and fragile.

The last option (in most respects, a hideous one, mentioned primarily to convince people that there must be a better way) is rubbing a “before” and an “after” image of the operational file together. In this option, a snapshot of a database is taken at the moment of extraction. When another extraction is performed, another snapshot is taken. The two snapshots are serially compared to each other to determine the activity that has transpired. This approach is cumbersome and complex, and it requires an inordinate amount of resources. It is simply a last resort to be done when nothing else works.

Integration and performance are not the only major discrepancies that prevent a simple extract process from being used to construct the data warehouse. A third difficulty is that operational data must undergo a time-basis shift as it passes into the data warehouse, as shown in Figure 3.5.

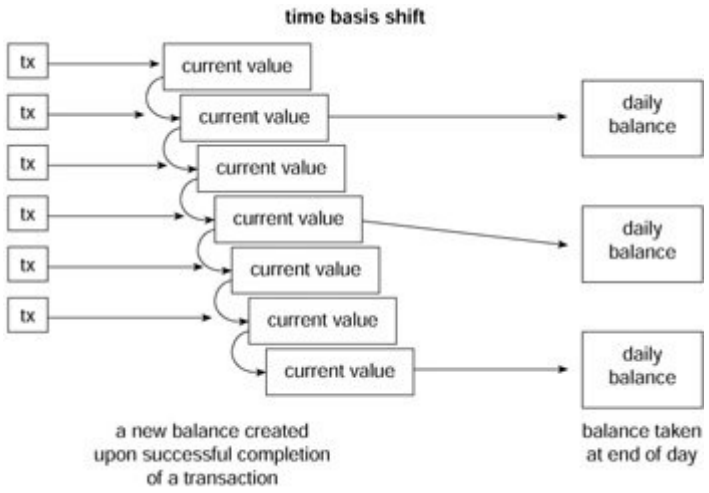

Figure 3.5: A shift in time basis is required as data is moved over from the operational to the data warehouse environment.

Existing operational data is almost always current-value data. Such data's accuracy is valid as of the moment of access, and it can be updated. But data that goes into the data warehouse cannot be updated. Instead, an element of time must be attached to it. A major shift in the modes of processing surrounding the data is necessary as it passes into the data warehouse from the operational environment.

Yet another major consideration when passing data is the need to manage the volume of data that resides in and passes into the warehouse. Data must be condensed both at the moment of extraction and as it arrives at the warehouse. If condensation is not done, the volume of data in the data warehouse will grow rapidly out of control. Figure 3.6 shows a simple form of data condensation.

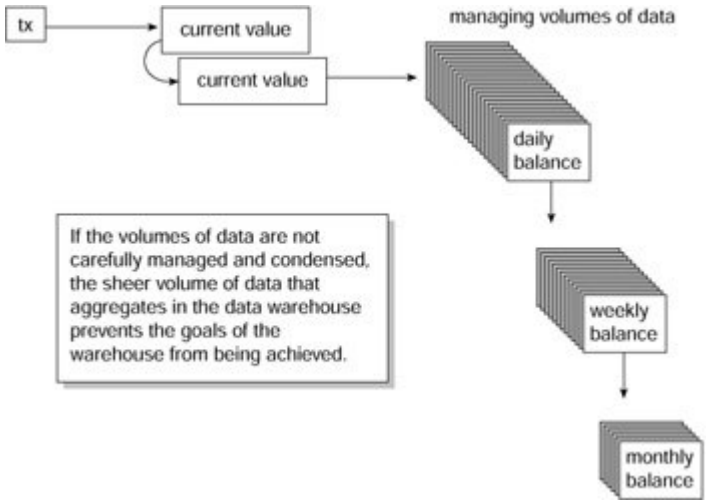

Figure 3.6: Condensation of data is a vital factor in the managing of warehouse data

Data/Process Models and the Architected Environment

Before attempting to apply conventional database design techniques, the designer must understand the applicability and the limitations of those techniques. Figure 3.7 shows the relationship among the levels of the architecture and the disciplines of process modeling and

data modeling. The process model applies only to the operational environment. The data model applies to both the operational environment and the data warehouse environment. Trying to use a process or data model in the wrong place produces nothing but frustration.

In general there are two types of models for the information systems environment—data models and process models. Data models are discussed in depth in the following section. For now, we will address process models. A process model typically consists of the following (in whole or in part):

- Functional decomposition
- Context-level zero diagram
- Data flow diagram
- Structure chart
- State transition diagram
- HIPO chart
- Pseudocode

There are many contexts and environments in which a process model is invaluable—for instance, when building the data mart. However, because the process model is requirements-based, it is not suitable for the data warehouse. The process model assumes that a set of known processing requirements exists—a priori—before the details of the design are established. With processes, such an assumption can be made. But those assumptions do not hold for the data warehouse. Many development tools, such as CASE tools, have the same orientation and as such are not applicable to the data warehouse environment.

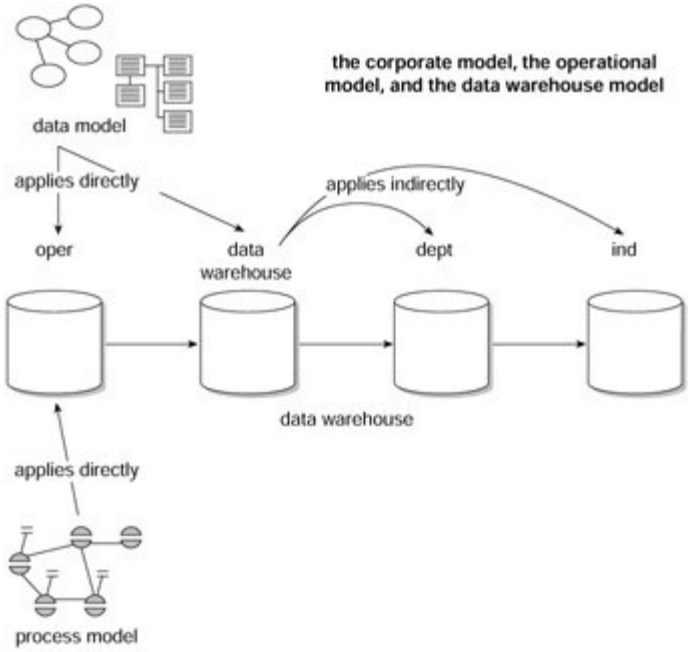

Figure 3.7: How the different types of models apply to the architected environment

The Data Warehouse and Data Models

As shown in Figure 3.8, the data model is applicable to both the existing systems environment and the data warehouse environment. Here, an overall corporate data model has been constructed with no regard for a distinction between existing operational systems and the data warehouse. The corporate data model focuses on and represents only primitive

data. To construct a separate existing data model, the beginning point is the corporate model, as shown. Performance factors are added into the corporate data model as the model is transported to the existing systems environment. All in all, very few changes are made to the corporate data model as it is used operationally.

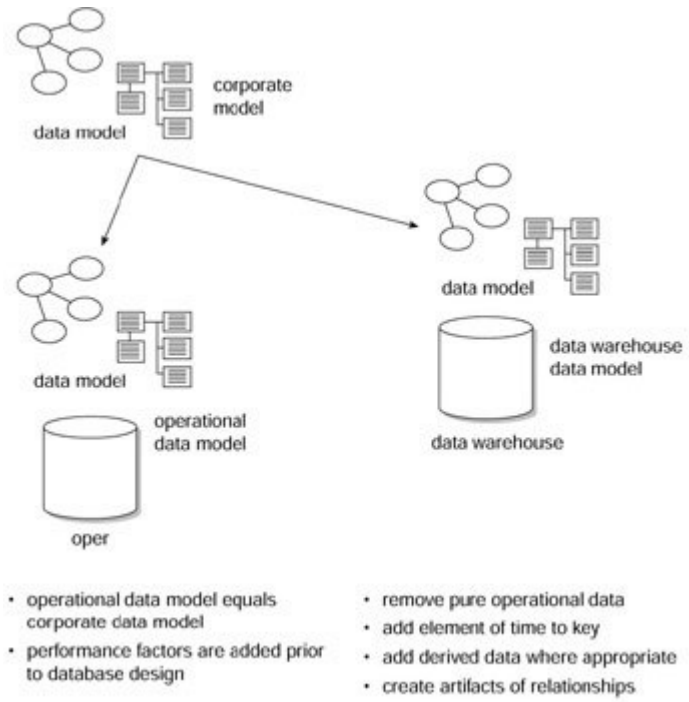

Figure 3.8: How the different levels of modeling relate.

However, a fair number of changes are made to the corporate data model asit is applied to the data warehouse. First, data that is used purely in the operational environment is removed. Next, the key structures of the corporate data model are enhanced with an element of time. Derived data is added to the corporate data model where the derived data is publicly used and calculated once, not repeatedly. Finally, data relationships in the operational environment are turned into “artifacts” in the data warehouse.

A final design activity in transforming the corporate data model to the data warehouse data model is to perform “stability” analysis. Stability analysis involves grouping attributes of data together based on their propensity for change. Figure 3.9 illustrates stability analysis for the manufacturing environment. Three tables are created from one large general-purpose table based on the stability requirements of the data contained in the tables.

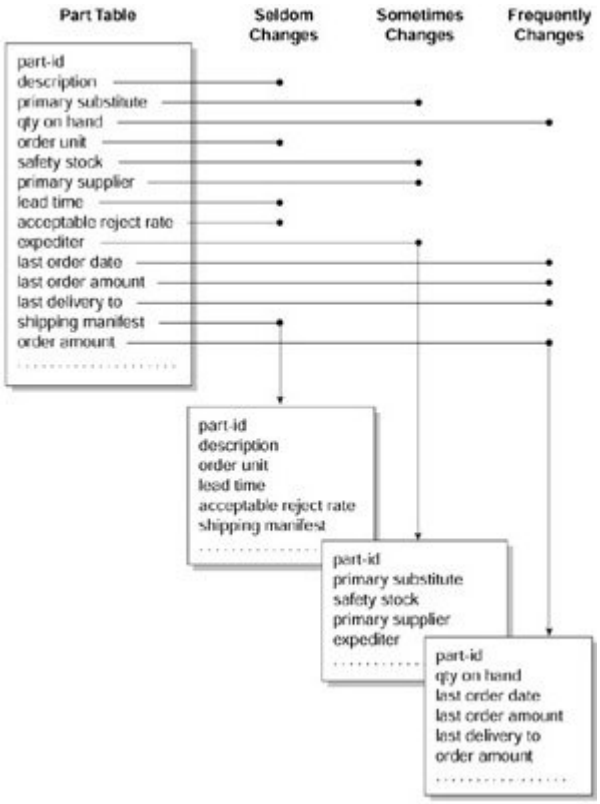

Figure 3.9: An example of stability analysis.

In Figure 3.9, data that seldom changes is grouped, data that sometimes changes is grouped, and data that frequently changes is grouped. The net result of stability analysis (which usually is the last step of data modeling before physical database design) is to create groups of data with similar characteristics.

There is, then, a common genesis of data models. As an analogy, say the corporate data model is Adam, the operational data model is Cain, and the data warehouse data model is Abel. They are all from the same lineage, but at the same time, they are all different.

The Data Warehouse Data Model

(Other books have been written on data modeling, detailing several different approaches. Any number can be used successfully in building a data warehouse. The approach summarized here can be further explored in my previous book *Information Systems Architecture: Development in the 90s* [QED/Wiley, 1992].)

There are three levels of data modeling: high-level modeling (called the ERD, entity relationship level), midlevel modeling (called the data item set, or DIS), and low-level modeling (called the physical model).

The high level of modeling features entities and relationships, as shown in Figure 3.10. The name of the entity is surrounded by an oval. Relationships among entities are depicted with arrows. The direction and number of the arrowheads indicate the cardinality of the relationship, and only direct relationships are indicated. In doing so, transitive dependencies are minimized.

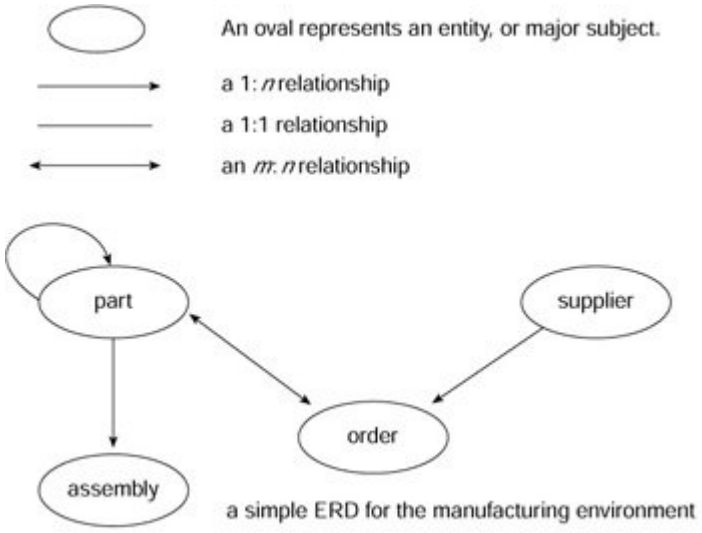

Figure 3.10: Representing entities and relationships.

The entities that are shown in the ERD level are at the highest level of abstraction. What entities belong in the scope of the model and what entities do not are determined by what is termed the “scope of integration,” as shown in Figure 3.11. The scope of integration defines the boundaries of the data model and needs to be defined before the modeling process commences. The scope is agreed on by the modeler, the management, and the ultimate user of the system. If the scope is not predetermined, there is the great chance that the modeling process will continue forever. The definition of the scope of integration should be written in no more than five pages and in language understandable to the businessperson.

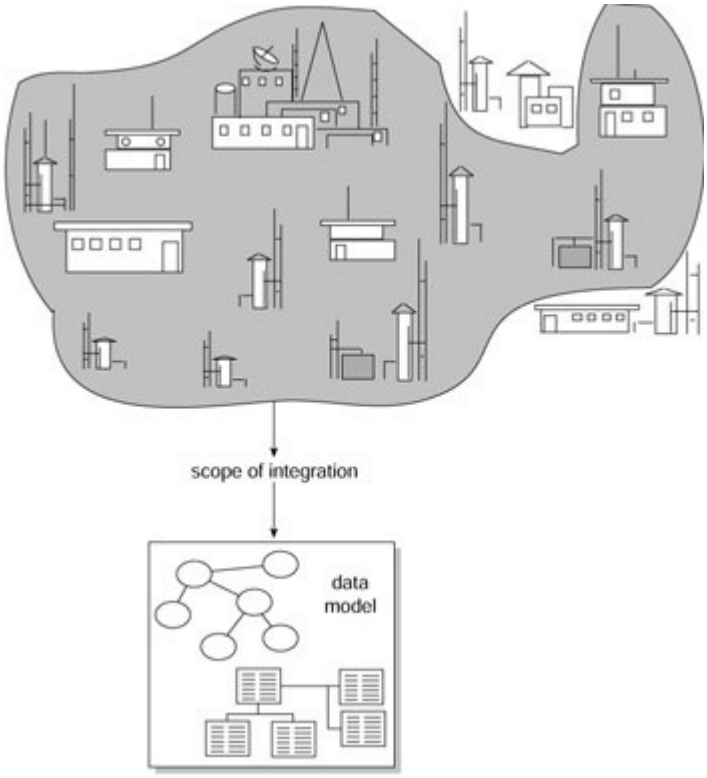

Figure 3.11: The scope of integration determines what portion of the enterprise will be reflected in the data model.

As shown in Figure 3.12, the corporate ERD is a composite of many individual ERDs that reflect the different views of people across the corporation. Separate high-level data models have been created for different communities within the corporation. Collectively, they make up the corporate ERD.

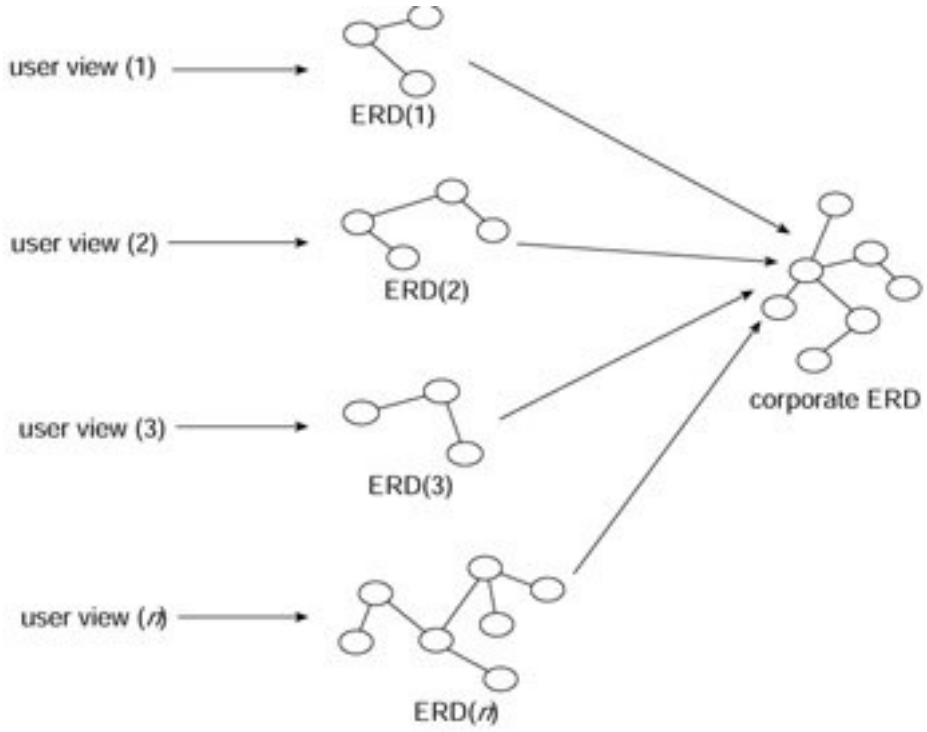

Figure 3.12: The construction of the corporate ERD from the different user view ERDs.

The ERDs representing the known requirements of the DSS community are created by means of user view sessions, which are interview sessions with the appropriate personnel in the various departments.

The Midlevel Data Model

After the high-level data model is created, the next level is established—the midlevel model, or the DIS. For each major subject area, or entity, identified in the high-level data model, a midlevel model is created, as seen in Figure 3.13. The high-level data model has identified four entities, or major subject areas. Each area is subsequently developed into its own midlevel model.

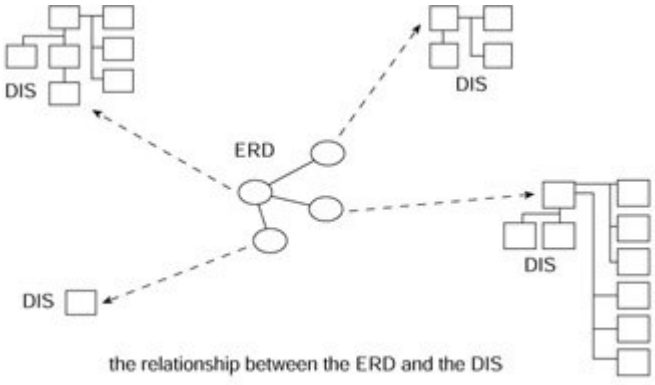

Figure 3.13: Each entity in the ERD is further defined by its own DIS.

Interestingly, only very rarely are all of the midlevel models developed at once. The midlevel data model for one major subject area is expanded, then a portion of the model is fleshed out while other parts remain static, and so forth.

Shown in Figure 3.14, four basic constructs are found at the midlevel model:

- A primary grouping of data
- A secondary grouping of data
- A connector, signifying the relationships of data between major subject areas
- "Type of" data

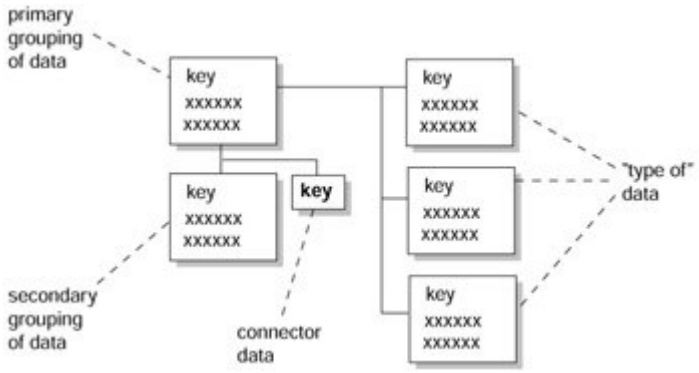

Figure 3.14: The four constructs that make up the midlevel data model.

The primary grouping exists once, and only once, for each major subject area. It holds attributes that exist only once for each major subject area. As with all groupings of data, the primary grouping contains attributes and keys for each major subject area.

The secondary grouping holds data attributes that can exist multiple times for each major subject area. This grouping is indicated by a line emanating downward from the primary grouping of data. There may be as many secondary groupings as there are distinct groups of data that can occur multiple times.

The third construct is the connector. The connector relates data from one grouping to another. A relationship identified at the ERD level results in an acknowledgment at the DIS level. The convention used to indicate a connector is an underlining of a foreign key.

The fourth construct in the data model is "type of" data. "Type of" data is indicated by a line leading to the right of a grouping of data. The grouping of data to the left is the supertype. The grouping of data to the right is the subtype of data.

These four data modeling constructs are used to identify the attributes of data in a data model and the relationship among those attributes. When a relationship is identified at the ERD level, it is manifested by a pair of connector relationships at the DIS level. Figure 3.15 shows one of those pairs.

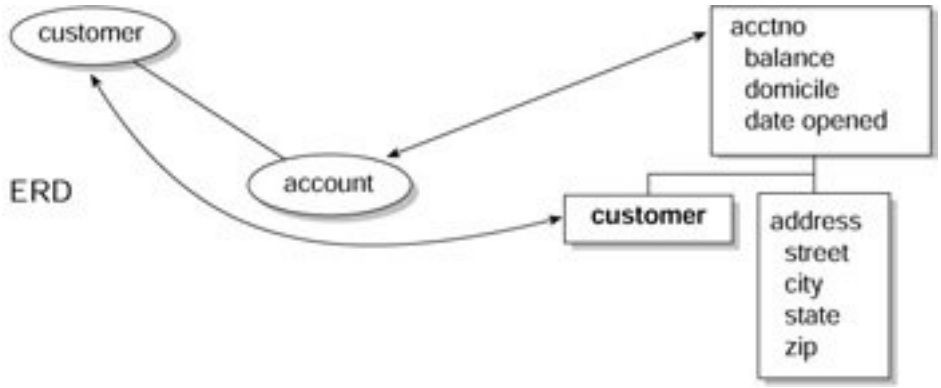

Figure 3.15: The relationships identified in the ERD are reflected by connectors in the DIS. Note that only one connector—from acctno to customer—is shown in this diagram. In reality, another connector from customer to acctno would be shown elsewhere in the DIS for customer.

At the ERD, a relationship between customer and account has been identified. At the DIS level for account, there exists a connector beneath account. This indicates that an account may have multiple customers attached to it. Not shown is the corresponding relationship beneath the customer in the customer DIS. Here will be a connector to account, indicating that a customer can have an account or accounts.

Figure 3.16 shows what a full-blown DIS might look like, where a DIS exists for an account for a financial institution. In this example, all of the different constructs are shown in the DIS.

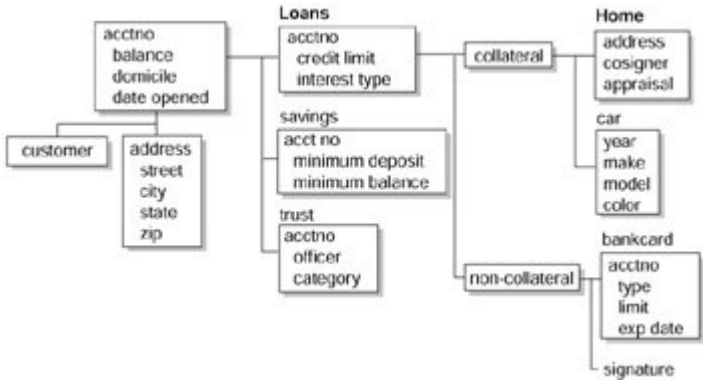

Figure 3.16: An expanded DIS showing the different types of loans that a bank may support.

Of particular interest is the case where a grouping of data has two “type of” lines emanating from it, as seen in Figure 3.17. The two lines leading to the right indicate that there are two “type of” criteria. One type of criteria is by activity type—either a deposit or a withdrawal. The other line indicates another—either an ATM activity or a teller activity. Collectively, the two types of activity encompass the following transactions:

- ATM deposit
- ATM withdrawal
- Teller deposit
- Teller withdrawal

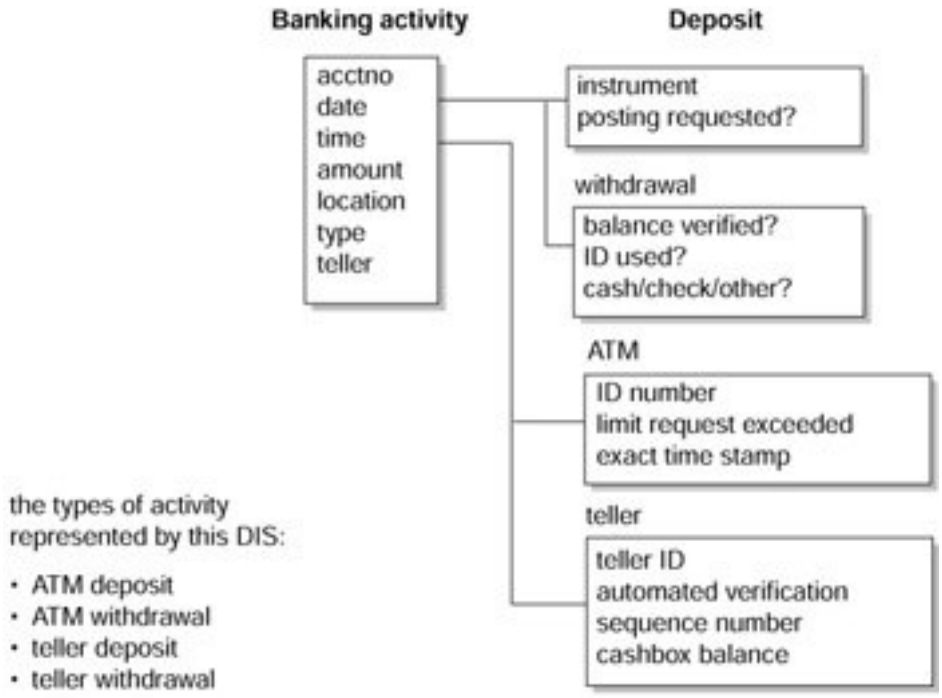

Figure 3.17: A DIS showing different subcategorization criteria.

Another feature of the diagram is that all common data is to the left and all unique data is to the right. For example, the attributes date and time are common to all transactions, but cashbox balance relates only to teller activity.

The relationship between the data model and the physical tables that result from the data model is shown in Figure 3.18. In general, each grouping of data results in a table being defined in the database design process. Assuming that to be the case, two transactions result in several table entries, as seen in the Figure 3.18. The physical table entries that resulted came from the following two transactions:

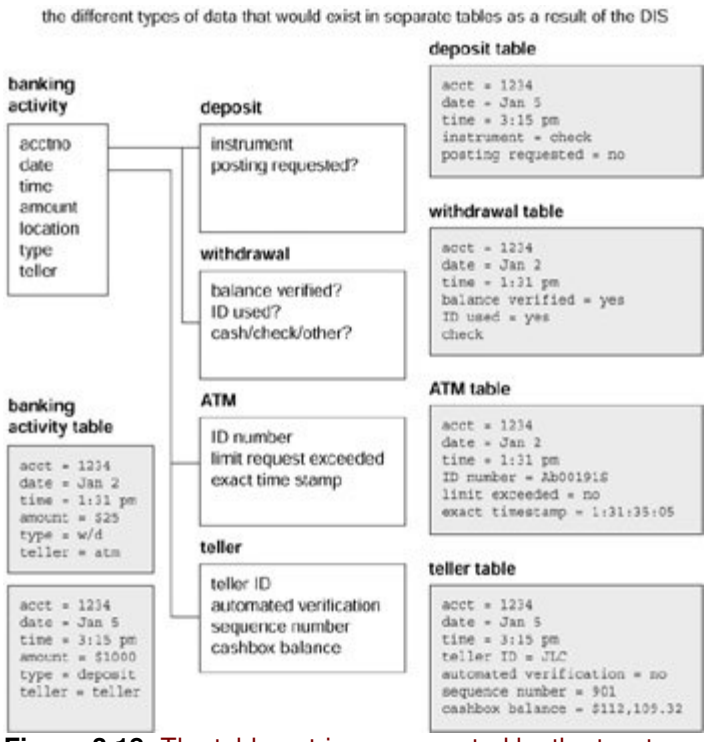

Figure 3.18: The table entries represented by the two transactions.

- An ATM withdrawal that occurred at 1:31 P.M. on January 2
- A teller deposit that occurred at 3:15 P.M. on January 5

The two transactions caused six entries in five separate tables to be generated.

Like the corporate ERD that is created from different ERDs reflecting the community of users, the corporate DIS is created from multiple DISs, as illustrated by Figure 3.19. When the interviews or JAD sessions are done for a particular user, a DIS, as well as an ERD, is created. The parochial DIS is then merged with all other DISs to form the corporate view of a DIS.

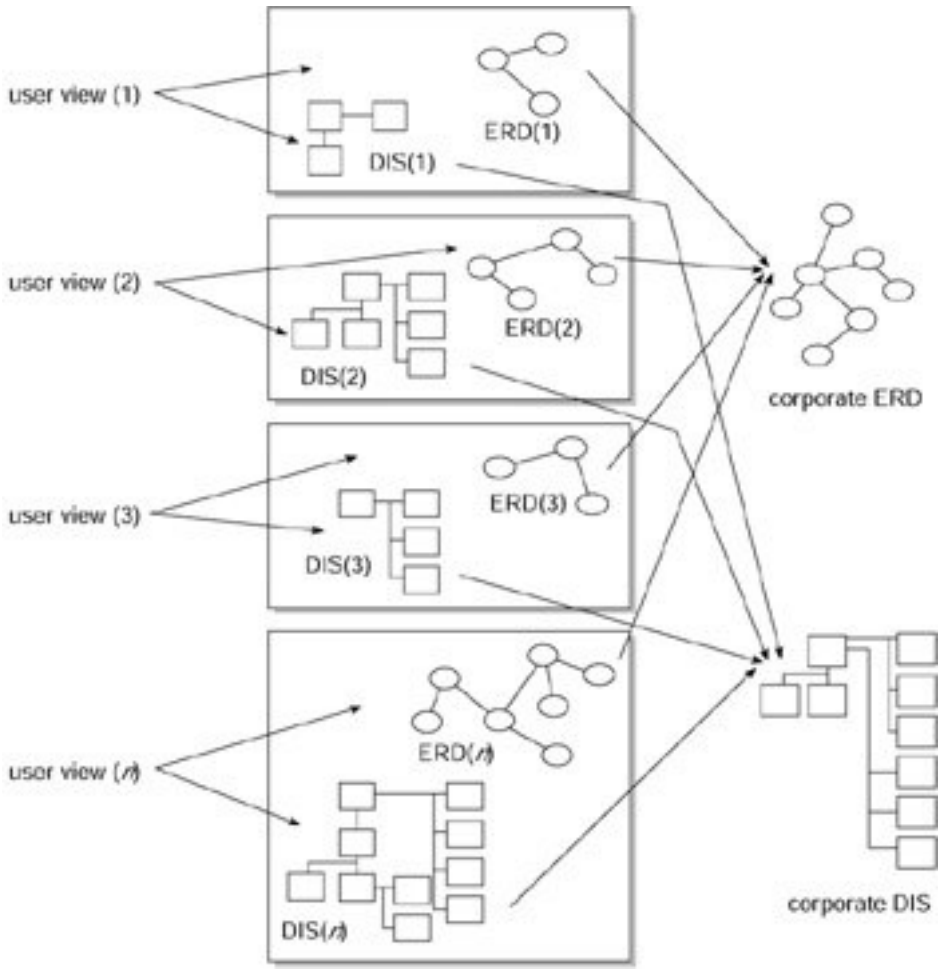

Figure 3.19: The corporate DIS is made up of the DIS created as a result of each user view session.

The Physical Data Model

The physical data model is created from the midlevel data model merely by extending the midlevel data model to include keys and physical characteristics of the model. At this point, the physical data model looks like a series of tables, sometimes called relational tables.

Although it is tempting to say that the tables are ready to be cast into the concrete of physical database design, one last design step remains—factoring in the performance characteristics. With the data warehouse, the first step in doing so is deciding on the granularity and partitioning of the data. This is crucial. (Of course, the key structure is changed to add the element of time, to which each unit of data is relevant.)

After granularity and partitioning are factored in, a variety of other physical design activities are embedded into the design, as outlined in Figure 3.20. At the heart of the physical design considerations is the usage of physical I/O (input/output). Physical I/O is the activity that brings data into the computer from storage or sends data to storage from the computer. Figure 3.21 shows a simple case of I/O.

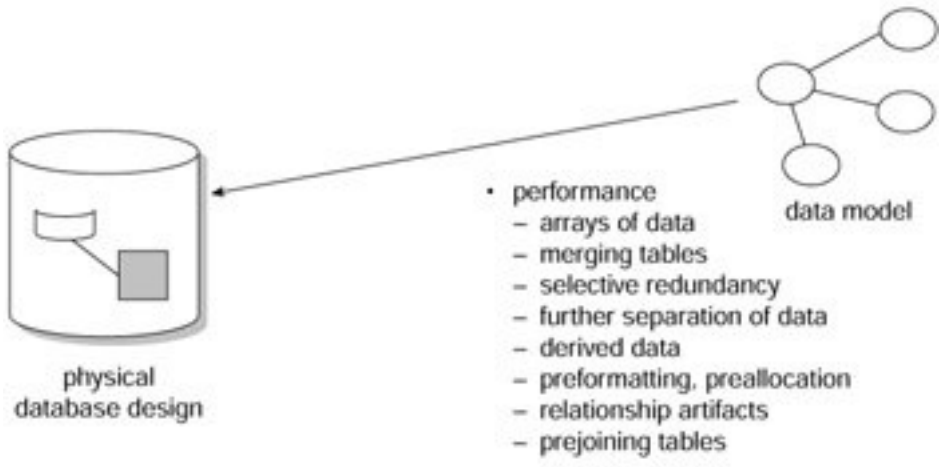

Figure 3.20: Getting good performance out of the data warehouse environment.

Data is transferred to and from the computer to storage in blocks. The I/O event is vital to performance because the transfer of data to and from storage to the computer occurs roughly two to three orders of magnitude slower than the speeds at which the computer runs. The computer runs internally in terms of nanosecond speed. Transfer of data to and from storage occurs in terms of milliseconds. Thus, physical I/O is the main impediment to performance.

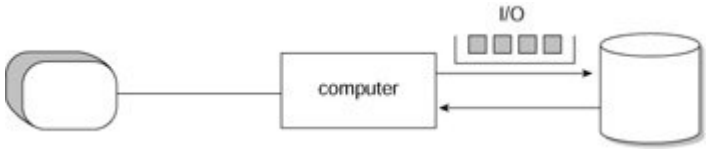

Figure 3.21: Getting the most out of the physical I/Os that have to be done.

The job of the data warehouse designer is to organize data physically for the return of the maximum number of records from the execution of a physical I/O. (Note: This is not an issue of blindly transferring a large number of records from DASD to main storage; instead, it is a more sophisticated issue of transferring a bulk of records that have a high probability of being accessed.)

For example, suppose a programmer must fetch five records. If those records are organized into different blocks of data on storage, then five I/Os will be required. But if the designer can anticipate that the records will be needed as a group and can physically juxtapose those records into the same block, then only one I/O will be required, thus making the program run much more efficiently.

There is another mitigating factor regarding physical placement of data in the data warehouse: Data in the warehouse normally is not updated. This frees the designer to use physical design techniques that otherwise would not be acceptable if it were regularly updated.

The Data Model and Iterative Development

In all cases, the data warehouse is best built iteratively. The following are some of the many reasons why iterative development is important:

- The industry track record of success strongly suggests it.

- The end user is unable to articulate requirements until the first iteration is done.
- Management will not make a full commitment until actual results are tangible and obvious.
- Visible results must be seen quickly.

What may not be obvious is the role of the data model in iterative development. To understand the role of the data model during this type of development, consider the typical iterative development suggested by Figure 3.22. First, one development effort is undertaken, then another, and so forth. The data warehouse serves as a roadmap for each of the development efforts, as seen in Figure 3.23.

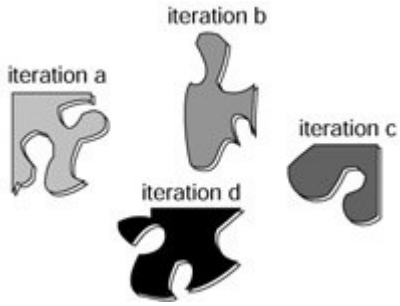

Figure 3.22: The different iterations of data warehouse development.

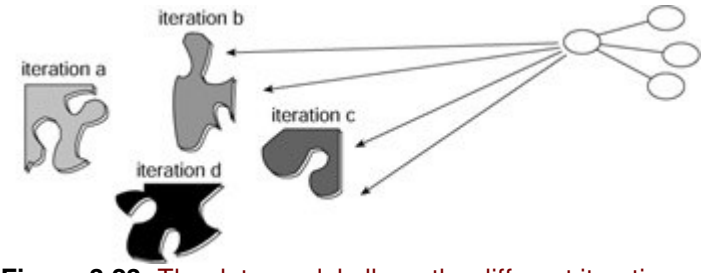

Figure 3.23: The data model allows the different iterations of development to be built in a cohesive manner.

When the second development effort ensues, the developer is confident that he or she will intersect his or her effort with the first development effort because all development efforts are being driven from the data model. Each succeeding development effort builds on the preceding one. The result is that the different development efforts are done under a unifying data model. And because they are built under a single data model, the individual iterative efforts produce a cohesive and tightly orchestrated whole, as seen in Figure 3.24.

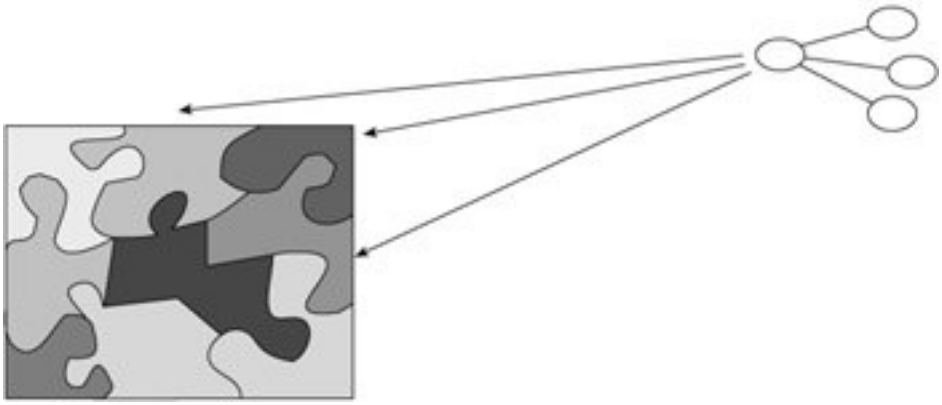

Figure 3.24: At the end of the development effort, all the iterations fit together.

When the different iterations of development are done with no unifying data model, there is much overlap of effort and much separate, disjoint development. Figure 3.25 suggests this cacophonous result.

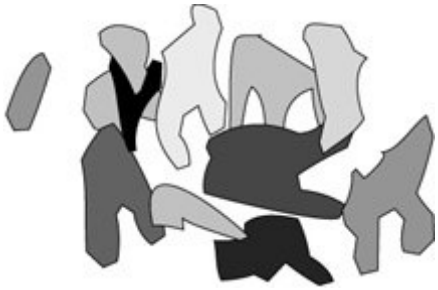

Figure 3.25: When there is no data model, the iterations do not form a cohesive pattern. There is much overlap and lack of uniformity.

There is, then, an indirect, yet important, correlation between the data model and the ability to achieve long-term integration and a harmonious effort in the incremental and iterative development of a data warehouse

Normalization/Denormalization

The output of the data model process is a series of tables, each of which contains keys and attributes. The normal output produces numerous table, each

with only a modicum of data. While there is nothing wrong—per se—with lots of little tables, there is a problem from a performance perspective. Consider the work the program has to do in order to interconnect the tables dynamically, as shown in Figure 3.26.

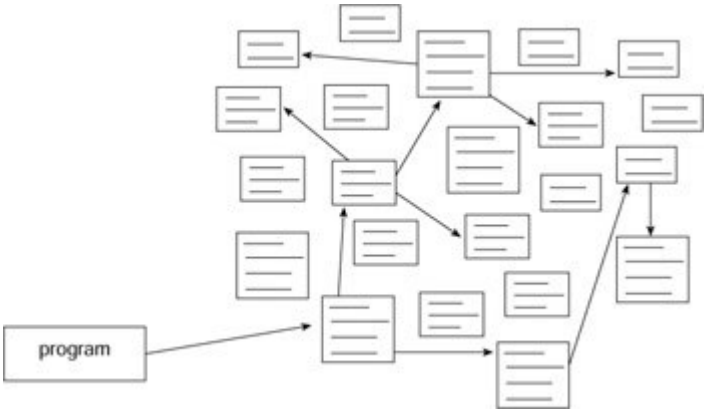

Figure 3.26: When there are many tables, much I/O is required for dynamic interconnectability.

In Figure 3.26, a program goes into execution. First, one table is accessed, then another. To execute successfully, the program must jump around many tables. Each time the program jumps from one table to the next, I/O is consumed, in terms of both accessing the data and accessing the index to find the data. If only one or two programs had to pay the price of I/O, there would be no problem. But when all programs must pay a stiff price for I/O, performance in general suffers, and that is precisely what happens when many small tables, each containing a limited amount of data, are created as a physical design.

A more rational approach is to physically merge the tables so that minimal I/O is consumed, as seen in Figure 3.27. Now the same program operates as before, only it needs much less I/O to accomplish the same task.

Figure 3.27: When tables are physically merged, much less I/O is required.

The question, then, becomes what is a sane strategy to merge the tables so that the maximum benefit is derived? It is in answering this question that the physical database designer earns his or her reward.

Merging tables is only one design technique that can save I/O. Another very useful technique is creating an array of data. In Figure 3.28, data is normalized so that each occurrence of a sequence of data resides in a different physical location. Retrieving each occurrence, *n*, *n* 1 1, *n* 1 2, ... , requires a physical I/O to get the data. If the data were placed in a single row in an array, then a single I/O would suffice to retrieve it, as shown at the bottom of Figure 3.28.

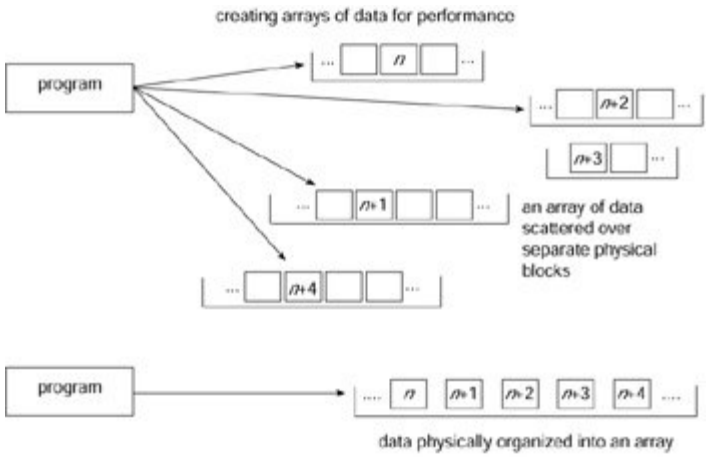

Figure 3.28: Under the right circumstances, creating arrays of data can save considerable resources.

Of course, it does not make sense to create an array of data in every case. Only when there are a stable number of occurrences, where the data is accessed in sequence, where it is created and/or updated in a statistically well-behaved sequence, and so forth, does creating an array pay off.

Interestingly, in the data warehouse these circumstances occur regularly because of the time-based orientation of the data. Data warehouse data is always relevant to some moment in time, and units of time occur with great regularity. In the data warehouse, creating an array by month, for example, is a very easy, natural thing to do.

Another important design technique that is especially relevant to the data warehouse environment is the deliberate introduction of redundant data. Figure 3.29 shows an example where the deliberate introduction of redundant data pays a big dividend. In the top of Figure 3.29, the field—description—is normalized and exists nonredundantly. In doing so, all processes that must see the description of a part must access the base parts table. The access of the data is very expensive, although the update of the data is optimal.

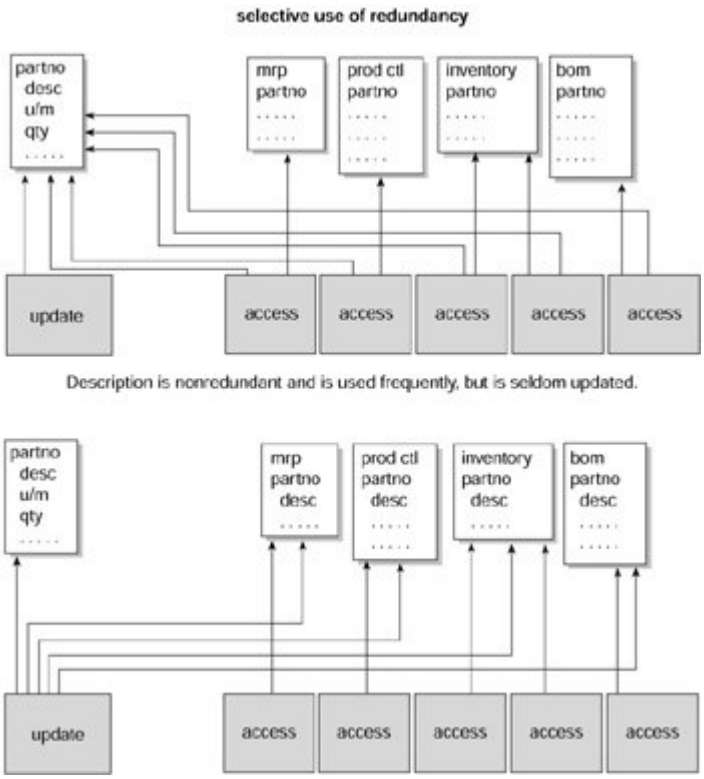

Figure 3.29: Description is redundantly spread over the many places it is used. It must be updated in many places when it changes, but it seldom, if ever, does.

In the bottom of Figure 3.29, the data element—description—has been deliberately placed in the many tables where it is likely to be used. In doing so, the access of data is more efficient, and the update of data is not optimal. For data that is widely used (such as description), and for data that is stable (such as description), however, there is little reason to worry about update. In particular, in the data warehouse environment there is no concern whatsoever for update.

Another useful technique is the further separation of data when there is a wide disparity in the probability of access. Figure 3.30 shows such a case.

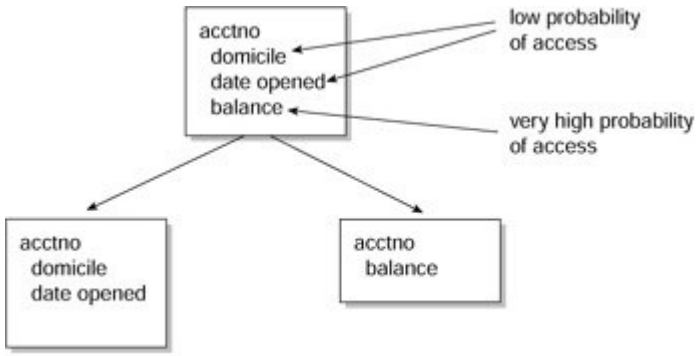

Figure 3.30: Further separation of data based on a wide disparity in the probability of access.

In Figure 3.30, concerning a bank account, the domicile of the account and the data opened for the account are normalized together with the balance of the account. Yet the balance of

the account has a very different probability of access than the other two data elements. The balance of an account is very popular, while the other data is hardly ever accessed. To make I/O more efficient and to store the data more compactly, it makes sense to further reduce the normalized table into two separate tables, as shown.

Occasionally, the introduction of derived (i.e., calculated) data into the physical database design can reduce the amount of I/O needed. Figure 3.31 shows such a case. A program accesses payroll data regularly in order to calculate the annual pay and taxes that have been paid. If the program is run regularly and at the year's end, it makes sense to create fields of data to store the calculated data. The data has to be calculated only once. Then all future requirements can access the calculated field. This approach has another advantage in that once the field is calculated, it will not have to be calculated again, eliminating the risk of faulty algorithms from incorrect evaluations.

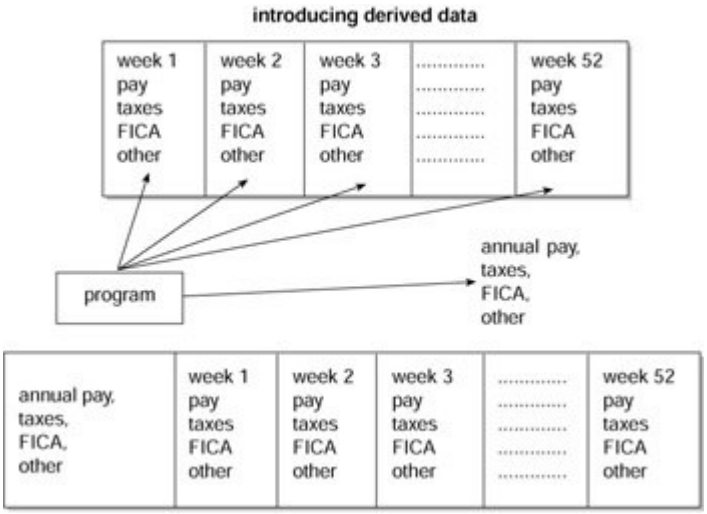

Figure 3.31: Derived data, calculated once, then forever available.

One of the most innovative techniques in building a data warehouse is what can be termed a "creative" index, or a creative profile (a term coined by Les Moore). Figure 3.32 shows an example of a creative index. This type of creative index is created as data is passed from the operational environment to the data warehouse environment. Because each unit of data has to be handled in any case, it requires very little overhead to calculate or create an index at this point.

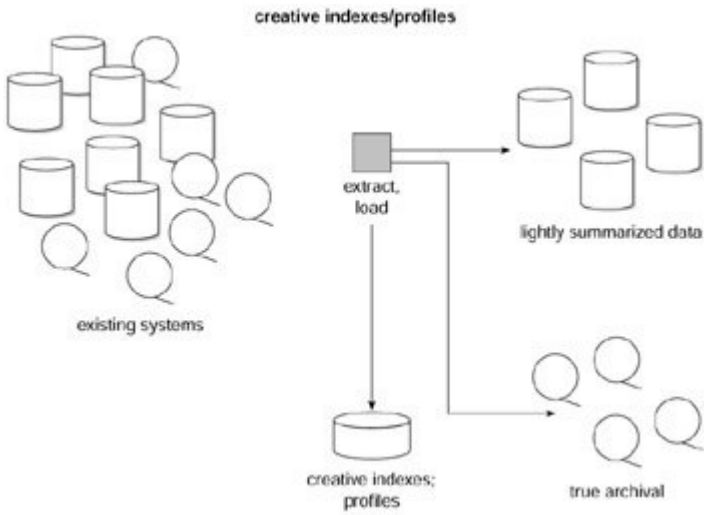

Figure 3.32: Examples of creative indexes:

- The top 10 customers in volume are
- The average transaction value for this extract was $nnn.nn
- The largest transaction was $nnn.nn.
- The number of customers who showed activity without purchasing was nn.

The creative index does a profile on items of interest to the end user, such as the largest purchases, the most inactive accounts, the latest shipments, and so on. If the requirements that might be of interest to management can be anticipated (admittedly, they cannot in every case), at the time of passing data to the data warehouse, it makes sense to build a creative index.

A final technique that the data warehouse designer should keep in mind is the management of referential integrity. Figure 3.33 shows that referential integrity appears as “artifacts” of relationships in the data warehouse environment.

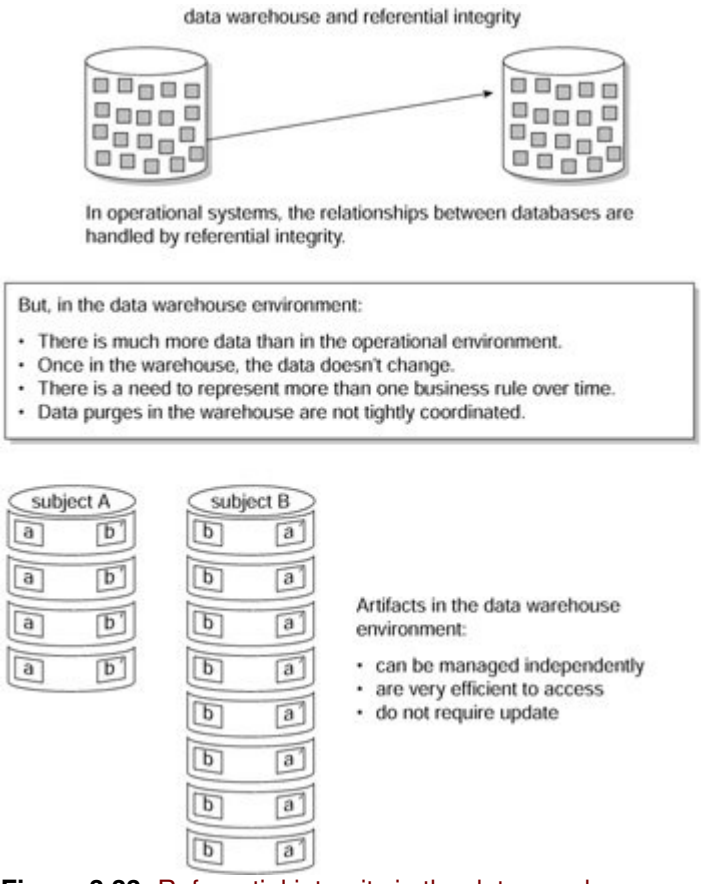

Figure 3.33: Referential integrity in the data warehouse environment.

In the operational environment, referential integrity appears as a dynamic link among tables of data. But because of the volume of data in a data warehouse, because the data warehouse is not updated, and because the warehouse represents data over time and relationships do not remain static, a different approach should be taken toward referential integrity. In other words, relationships of data are represented by an artifact in the data warehouse environment. Therefore, some data will be duplicated, and some data will be deleted when other data is still in the warehouse. In any case, trying to replicate referential integrity in the data warehouse environment is a patently incorrect approach.

Snapshots in the Data Warehouse

Data warehouses are built for a wide variety of applications and users, such as customer systems, marketing systems, sales systems, and quality control systems. Despite the very diverse applications and types of data warehouses, a common thread runs through all of them. Internally, each of the data warehouses centers around a structure of data called a "snapshot." Figure 3.34 shows the basic components of a data warehouse snapshot.

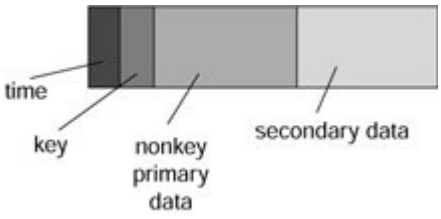

Figure 3.34: A data warehouse record of data is a snapshot taken at one moment in time and includes a variety of types of data.

Snapshots are created as a result of some event occurring. Several kinds of events can trigger a snapshot. One event is the recording of information about a discrete activity, such writing a check, placing a phone call, the receipt of a shipment, the completion of an order, or the purchase of a policy. In the case of a discrete activity, some business occurrence has occurred, and the business must make note of it. In general, discrete activities occur randomly.

The other type of snapshot trigger is time, which is a predictable trigger, such as the end of the day, the end of the week, or the end of the month.

The snapshot triggered by an event has four basic components:

- A key
- A unit of time
- Primary data that relates only to the key
- Secondary data captured as part of the snapshot process that has no direct relationship to the primary data or key

Of these components, only secondary data is optional.

The key can be unique or nonunique and it can be a single element of data. In a typical data warehouse, however, the key is a composite made up of many elements of data that serve to identify the primary data. The key identifies the record and the primary data.

The unit of time, such as year, month, day, hour, and quarter, usually (but not always) refers to the moment when the event being described by the snapshot has occurred. Occasionally, the unit of time refers to the moment when the capture of data takes place. (In some cases a distinction is made between when an event occurs and when the information about the event is captured. In other cases no distinction is made.) In the case of events triggered by the passage of time, the time element may be implied rather than directly attached to the snapshot.

The primary data is the nonkey data that relates directly to the key of the record. As an example, suppose the key identifies the sale of a product. The element of time describes when the sale was finalized. The primary data describes what product was sold at what price, conditions of the sale, location of the sale, and who were the representative parties.

The secondary data—if it exists—identifies other extraneous information captured at the moment when the snapshot record was created. An example of secondary data that relates to a sale is incidental information about the product being sold (such as how much is in stock at the moment of sale). Other secondary information might be the prevailing interest rate for a bank's preferred customers at the moment of sale. Any incidental information can be added to a data warehouse record, if it appears at a later time that the information can be used for DSS processing. Note that the incidental information added to the snapshot may or may not be a foreign key. A foreign key is an attribute found in a table that is a reference to the key value of another table where there is a business relationship between the data found in the two tables.

Once the secondary information is added to the snapshot, a relationship between the primary and secondary information can be inferred, as shown in Figure 3.35. The snapshot implies that there is a relationship between secondary and primary data. Nothing other than the existence of the relationship is implied, and the relationship is implied only as of the instant of the snapshot. Nevertheless, by the juxtaposition of secondary and primary data in a snapshot record, at the instant the snapshot was taken, there is an inferred relationship of data. Sometimes this inferred relationship is called an "artifact." The snapshot record that has been described is the most general and most widely found case of a record in a data warehouse.

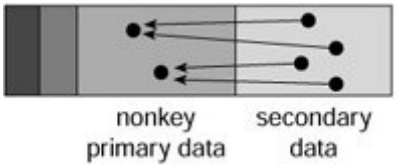

Figure 3.35: The artifacts of a relationship are captured as a result of the implied relationship of secondary data residing in the same snapshot as primary data

Meta Data

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

Meta data then acts like an index to the contents of the data warehouse. It sits above the warehouse and keeps track of what is where in the warehouse. Typically, items the meta data store tracks are as follows:

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

Managing Reference Tables in a Data Warehouse

When most people think of data warehousing, their thoughts turn to the normal, large databases constantly being used by organizations to run day-to-day business such as customer files, sales files, and so forth. Indeed, these common files form the backbone of the data warehousing effort. Yet another type of data belongs in the data warehouse and is often ignored: reference data.

Reference tables are often taken for granted, and that creates a special problem. For example, suppose in 1995 a company has some reference tables and starts to create its

data warehouse. Time passes, and much data is loaded into the data warehouse. In the meantime, the reference table is used operationally and occasionally changes. In 1999, the company needs to consult the reference table. A reference is made from 1995 data to the reference table. But the reference table has not been kept historically accurate, and the reference from 1995 data warehouse data to reference entries accurate as of 1999 produces very inaccurate results. For this reason, reference data should be made time-variant, just like all other parts of the data warehouse.

Reference data is particularly applicable to the data warehouse environment because it helps reduce the volume of data significantly. There are many design techniques for the management of reference data. Two techniques—at the opposite ends of the spectrum—are discussed here. In addition, there are many variations on these options.

Figure 3.36 shows the first design option, where a snapshot of an entire reference table is taken every six months This approach is quite simple and at first glance appears to make sense. But the approach is logically incomplete. For example, suppose some activity had occurred to the reference table on March 15. Say a new entry—ddw—was added, then on May 10 the entry for ddw was deleted. Taking a snapshot every six months would not capture the activity that transpired from March 15 to May 10.

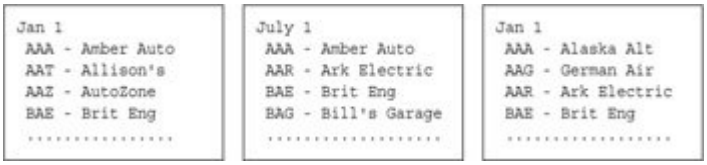

Figure 3.36: A snapshot is taken of a reference table in its entirety every six months—one approach to the management of reference tables in the data warehouse.

A second approach is shown in Figure 3.37. At some starting point a snapshot is made of a reference table. Throughout the year, all the activities against the reference table are collected. To determine the status of a given entry to the reference table at a moment in time, the activity is reconstituted against the reference table. In such a manner, logical completeness of the table can be reconstructed for any moment in time. Such a reconstruction, however, is a not a trivial matter; it may represent a very large and complex task.

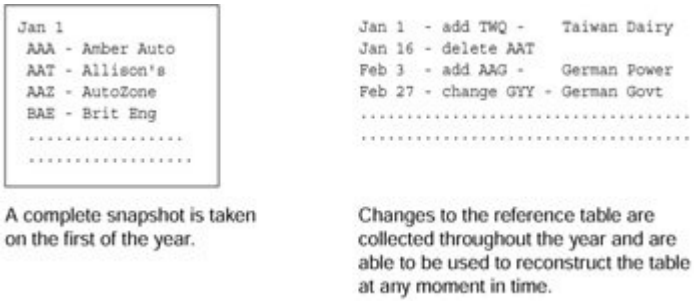

Figure 3.37: Another approach to the management of reference data.

The two approaches outlined here are opposite in intent. The first approach is simple but logically incomplete. The second approach is very complex but logically complete. Many design alternatives lie between these two extremes. However they are designed and implemented, reference tables need to be managed as a regular part of the data warehouse environment.

Cyclicity of Data—The Wrinkle of Time

One of the intriguing issues of data warehouse design is the cyclicity of data, or the length of time a change of data in the operational environment takes to be reflected in the data warehouse. Consider the data in Figure 3.38.

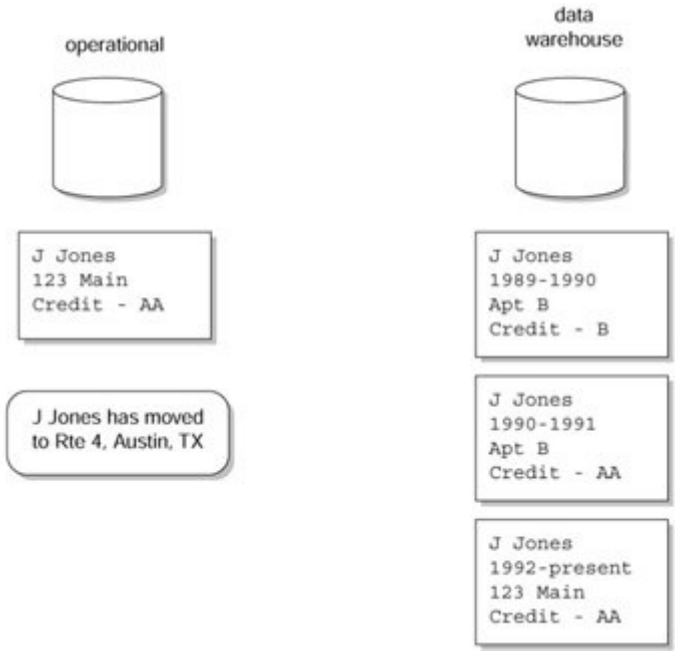

Figure 3.38: What happens when the corporation finds out that J Jones has moved?

The current information is shown for Judy Jones. The data warehouse contains the historical information about Judy. Now suppose Judy changes addresses. Figure 3.39 shows that as soon as that change is discovered, it is reflected in the operational environment as quickly as possible.

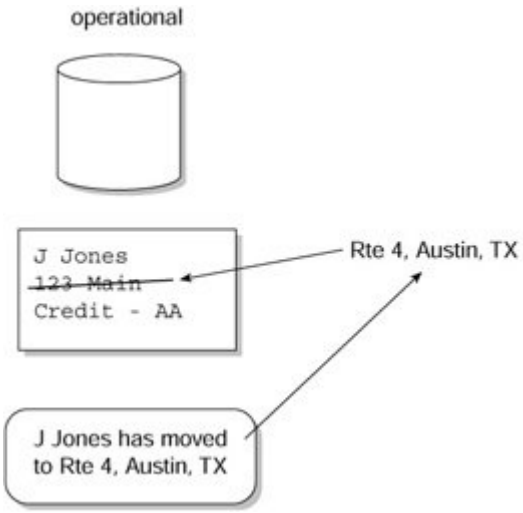

Figure 3.39: The first step is to change the operational address of J Jones.

Once the data is reflected in the operational environment, the changes need to be moved to the data warehouse. Figure 3.40 shows that the data warehouse has a correction to the ending date of the most current record and a new record has been inserted reflecting the change.

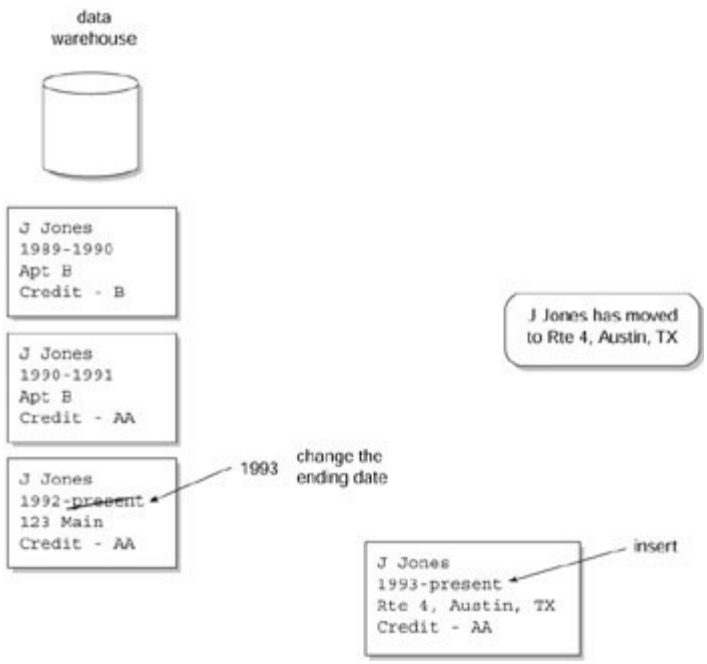

Figure 3.40: The activities that occur in the data warehouse as a result of the change of address.

The issue is, how soon should this adjustment to the data warehouse data be made? As a rule, at least 24 hours should pass from the time the change is known to the operational environment until the change is reflected into the data warehouse (see Figure 3.41). There should be no rush to try to move the change into the data warehouse as quickly as possible. This “wrinkle of time” should be implemented for several reasons.

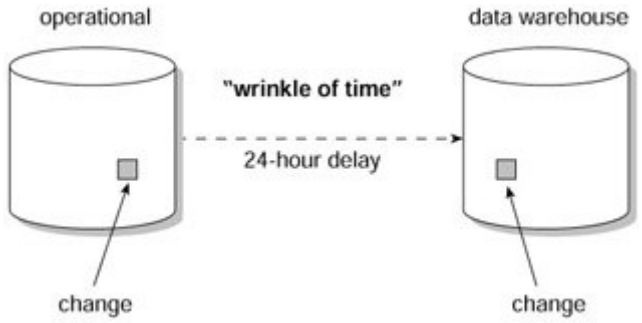

Figure 3.41: There needs to be at least a 24-hour lag—a “wrinkle of time”—between the time a change is known to the operational environment and the time when the change is reflected into the data warehouse.

The first reason is that the more tightly the operational environment is coupled to the data warehouse, the more expensive and complex the technology is. A 24-hour wrinkle of time can easily be accomplished with conventional technology. A 12-hour wrinkle of time can be

accomplished but at a greater cost of technology. A 6-hour wrinkle of time can be accomplished but at an even greater cost in technology.

A more powerful reason for the wrinkle of time is that it imposes a certain discipline on the environments. With a 24-hour wrinkle there is no temptation to do operational processing in the data warehouse and data warehouse processing in the operational environment. But if the wrinkle of time is reduced—say, to 4 hours—there is the temptation to do such processing, and that is patently a mistake.

Another benefit of the wrinkle of time is opportunity for data to settle before it is moved to the data warehouse. Adjustments can be made in the operational environment before the data is sent to the data warehouse. If data is quickly sent to the warehouse and then it is discovered that adjustments must be made, those adjustments need to be made in both the operational environment and the data warehouse environment.

Complexity of Transformation and Integration

At first glance, when data is moved from the legacy environment to the data warehouse environment, it appears that nothing more is going on than simple extraction of data from one place to the next. Because of the deceptive simplicity, many organizations start to build their data warehouses manually. The programmer looks at the movement of data from the old operational environment to the new data warehouse environment and declares “I can do that!” With pencil and coding pad in hand, the programmer anxiously jumps into the creation of code within the first three minutes of the design and development of the data warehouse.

First impressions, though, can be very deceiving. What at first appears to be nothing more than the movement of data from one place to another quickly turns into a large and complex task—far larger and more complex than the programmer thought.

Precisely what kind of functionality is required as data passes from the operational, legacy environment to the data warehouse environment? The following lists some of the necessary functionality:

- The extraction of data from the operational environment to the data warehouse environment requires a change in technology. This normally includes reading the operational DBMS technology, such as IMS, and writing the data out in newer, data warehouse DBMS technology, such as Informix. There is a need for a technology shift as the data is being moved. And the technology shift is not just one of a changing DBMS. The operating system changes, the hardware changes, and even the hardware-based structure of the data changes.
- The selection of data from the operational environment may be very complex. To qualify a record for extraction processing, several coordinated lookups to other records in a variety of other files may be necessary, requiring keyed reads, connecting logic, and so on. In some cases, the extraneous data cannot be read in anything but the online environment. When this is the case, extraction of data for the data warehouse must occur in the online operating window, a circumstance to be avoided if at all possible.
- Operational input keys usually need to be restructured and converted before they are written out to the data warehouse. Very seldom does an input key remain unaltered as it is read in the operational environment and written out to the data warehouse environment. In simple cases, an element of time is added to the output key structure. In complex cases, the entire input key must be rehashed or otherwise restructured.
- Nonkey data is reformatted as it passes from the operational environment to the data warehouse environment. As a simple example, input data about date is read as YYYY/MM/DD and is written to the output file as DD/MM/YYYY. (Reformatting of

operational data before it is ready to go into a data warehouse often becomes much more complex than this simple example.)

- Data is cleansed as it passes from the operational environment to the data warehouse environment. In some cases, a simple algorithm is applied to input data in order to make it correct. In complex cases, artificial intelligence subroutines are invoked to scrub input data into an acceptable output form. There are many forms of data cleansing, including domain checking, cross-record verification, and simple formatting verification.
- Multiple input sources of data exist and must be merged as they pass into the data warehouse. Under one set of conditions the source of a data warehouse data element is one file, and under another set of conditions the source of data for the data warehouse is another file. Logic must be spelled out to have the appropriate source of data contribute its data under the right set of conditions.
- When there are multiple input files, key resolution must be done before the files can be merged. This means that if different key structures are used in the different input files, the merging program must have the logic embedded that allows resolution.
- With multiple input files, the sequence of the files may not be the same or even compatible. In this case, the input files need resequenced. This is not a problem unless many records must be resequenced, which unfortunately is almost always the case.
- Multiple outputs may result. Data may be produced at different levels of summarization by the same data warehouse creation program.
- Default values must be supplied. Under some conditions an output value in the data warehouse will have no source of data. In this case, the default value to be used must be specified.
- The efficiency of selection of input data for extraction often becomes a real issue. Consider the case where at the moment of refreshment there is no way to distinguish operational data that needs to be extracted from operational data that does not need to be extracted. When this occurs, the entire operational file must be read. Reading the entire file is especially inefficient because only a fraction of the records is actually needed. This type of processing causes the online environment to be active, which further squeezes other processing in the online environment.
- Summarization of data is often required. Multiple operational input records are combined into a single “profile” data warehouse record. To do summarization, the detailed input records to be summarized must be properly sequenced. In the case where different record types contribute to the single summarized data warehouse record, the arrival of the different input record types must be coordinated so that a single record is produced.
- Renaming of data elements as they are moved from the operational environment to the data warehouse must be tracked. As a data element moves from the operational environment to the data warehouse environment, it usually changes its name. Documentation of that change must be made.
- The input records that must be read have exotic or nonstandard formats. There are a whole host of input types that must be read, then converted on entry into the data warehouse:
  - Fixed-length records
  - Variable-length records
  - Occurs depending on
  - Occurs clause

  Conversion must be made. But the logic of conversion must be specified, and the mechanics of conversion (what the “before” and “after” look like) can be quite complex. In some cases, conversion logic becomes very twisted.
- Perhaps the worst of all: Data relationships that have been built into old legacy program logic must be understood and unraveled before those files can be used as input. These relationships are often Byzantine, arcane, and undocumented. But they

must patiently be unwound and deciphered as the data moves into the data warehouse. This is especially difficult when there

- is no documentation or when the documentation that exists is out-of-date. And, unfortunately, on many operational legacy systems, there is no documentation. There is an old saying: Real programmers don't do documentation.
- Data format conversion must be done. EBCDIC to ASCII (or vice versa) must be spelled out.
- Massive volumes of input must be accounted for. Where there is only a small amount of data being entered as input, many design options can be accommodated. But where many records are being input, special design options (such as parallel loads and parallel reads) may have to be used.
- The design of the data warehouse must conform to a corporate data model. As such, there is order and discipline to the design and structuring of the data warehouse. The input to the data warehouse conforms to design specifications of an application that was written a long time ago. The business conditions behind the application have probably changed 10 times since the application was originally written. Much undocumented maintenance was done to the application code. In addition, the application probably had no integration requirements to fit with other applications. All of these mismatches must be accounted for in the design and building of the data warehouse.
- The data warehouse reflects the historical need for information, while the operational environment focuses on the immediate, current need for information. This means that an element of time may need to be added as the data moves from the operational environment to the data warehouse environment.
- The data warehouse addresses the informational needs of the corporation, while the operational environment addresses the up-to-the-second clerical needs of the corporation.
- Transmission of the newly created output file that will go into the data warehouse must be accounted for. In some cases, this is very easy to do; in other cases, it is not simple at all, especially when operating systems are crossed. Another issue is the location where the transformation will take place. Will the transformation take place on the machine hosting the operational environment? Or will raw data be transmitted and the transformation take place on the machine hosting the data warehouse?

And there is more. This list is merely a sampling of the complexities facing the programmer when setting off to load the data warehouse.

In the early days of data warehouse, there was no choice but to build the programs that did the integration by hand. Programmers using COBOL, C, and other languages wrote these. But soon people noticed that these programs were tedious and repetitive. Furthermore, these programs required ongoing maintenance. Soon technology appeared that automated the process of integrating data from the operational environment, called extract/transform/load (ETL) software. The first ETL software was crude, but it quickly matured to the point where almost any transformation could be handled.

ETL software comes in two varieties—software that produces code and software that produces a runtime module that is parameterized. The code producing software is much more powerful than the runtime software. The code producing software can access legacy data in its own format. The runtime software usually requires that legacy data be flattened. Once flattened, the runtime module can read the legacy data. Unfortunately, much intelligence is lost in the flattening of the legacy data.

In any case, ETL software automates the process of converting, reformatting, and integrating data from multiple legacy operational sources. Only under very unusual circumstances does attempting to build and maintain the operational/data warehouse interface manually make sense.

Triggering the Data Warehouse Record

The basic business interaction that causes the data warehouse to become populated with data is one that can be called an EVENT/SNAPSHOT interaction. In this type of interaction, some event (usually in the operational environment) triggers a snapshot of data, which in turn is moved to the data warehouse environment. Figure 3.42 symbolically depicts an EVENT/SNAPSHOT interaction.

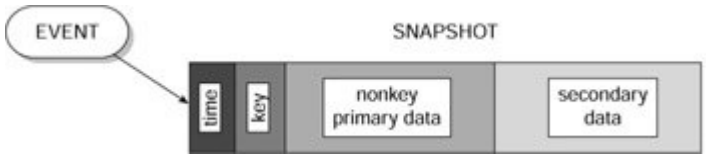

Figure 3.42: Every snapshot in the data warehouse is triggered by some event.

Events

As mentioned earlier in the chapter, the business event that triggers a snapshot might be the occurrence of some notable activity, such as the making of a sale, the stocking of an item, the placing of a phone call, or the delivery of a shipment. This type of business event is called an activity-generated event. The other type of business event that may trigger a snapshot is the marking of the regular passage of time, such as the ending of the day, the ending of the week, or the ending of the month. This type of business event is called a time-generated event.

Whereas events caused by business activities are random, events triggered by the passage of time are not. The time-related snapshots are created quite regularly and predictably.

Components of the Snapshot

Mentioned earlier in this chapter, the snapshot placed in the data warehouse normally contains several components. One component is the unit of time that marks the occurrence of the event. Usually (not necessarily always) the unit of time marks the moment of the taking of the snapshot. The next component of the snapshot is the key that identifies the snapshot. The third normal component of a data warehouse snapshot is the primary, nonkey data that relates to the key. Finally, an optional component of a snapshot is secondary data that has been incidentally captured as of the moment of the taking of the snapshot and placed in the snapshot. As mentioned, sometimes this secondary data is called an artifact of the relationship.

In the simplest case in a data warehouse, each operational activity important to the corporation will trigger a snapshot. In this case, there is a one-to-one correspondence between the business activities that have occurred and the number of snapshots that are placed in the data warehouse. When there is a one-to-one correspondence between the activities in the operational environment and the snapshots in the data warehouse, the data warehouse tracks the history of all the activity relating to a subject area.

Some Examples

An example of a simple snapshot being taken every time there is an operational, business activity might be found in a customer file. Every time a customer moves, changes phone numbers, or changes jobs, the data warehouse is alerted, and a continuous record of the history of the customer is made. One record tracks the customer from 1989 to 1991. The next record tracks the customer from 1991 to 1993. The next record tracks the customer from 1993 to the present. Each activity of the customer results in a new snapshot being placed in the data warehouse.

As another example, consider the premium payments on an insurance policy. Suppose premiums are paid semiannually. Every six months a snapshot record is created in the data warehouse describing the payment of the premium—when it was paid, how much, and so on.

Where there is little volume of data, where the data is stable (i.e., the data changes infrequently), and where there is a need for meticulous historical detail, the data warehouse can be used to track each occurrence of a business event by storing the details of every activity.

Profile Records

But there are many cases in which data in the data warehouse does not meet these criteria. In some cases, there will be massive volumes of data. In other cases, the content of data changes frequently. And in still other cases, there is no business need for meticulous historical detail of data. When one or more of these conditions prevail, a different kind of data warehouse record, called a profile or an aggregate record, can be created. A profile record groups many different, detailed occurrences of operational data into a single record. The single profile record represents the many operational records in aggregation.

Profile records represent snapshots of data, just like individual activity records. The difference between the two is that individual activity records in the data warehouse represent a single event, while profile records in the data warehouse represent multiple events.

Like individual activity records, profile records are triggered by some event-either a business activity or the marking of the regular passage of time. Figure 3.43 shows how an event causes the creation of a profile record.

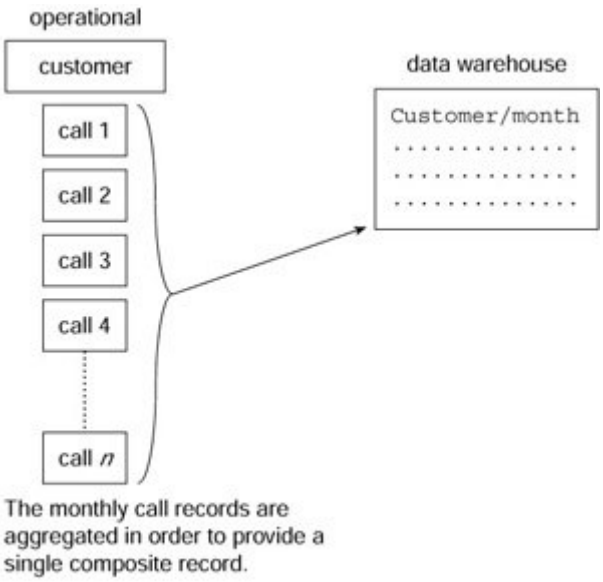

Figure 3.43: The creation of a profile record from a series of detailed records.

A profile record is created from the grouping of many detailed records. As an example, a phone company may at the end of the month take all of a customer's phone activities for the month and wrap those activities into a single customer record in the data warehouse. In doing so, a single representative record can be created for the customer that reflects all his or her monthly activity. Or a bank may take all the monthly activities of a customer and create an aggregate data warehouse record that represents all of his or her banking activities for the month.

The aggregation of operational data into a single data warehouse record may take many forms; for example:

- Values taken from operational data can be summarized.
- Units of operational data can be tallied, where the total number of units is captured.
- Units of data can be processed to find the highest, lowest, average, and so forth.
- First and last occurrences of data can be trapped.
- Data of certain types, falling within the boundaries of several parameters, can be measured.
- Data that is effective as of some moment in time can be trapped.
- The oldest and the youngest data can be trapped.

The ways to perform representative aggregation of operational data are limitless.

Another very appealing benefit to the creation of profile records is organizing the data in a compact and convenient form for the end user to access and analyze. Done properly, the end user is quite comfortable with the distillation of many records into a single record because he or she has to look only in a single place to find what is needed. By prepackaging the data into an aggregate record in the data warehouse, the data architect saves the end user from tedious processing.

Managing Volume

In many cases, the volume of data to be managed in the data warehouse is a significant issue. Creating profile records is an effective technique for managing the volume of data.

The reduction of the volume of data possible in moving detailed records in the operational environment into a profile record is remarkable. It is possible (indeed, normal) to achieve a 2-to-3 order-of-magnitude reduction of data by the creation of profile records in a data warehouse. Because of this benefit, the ability to create profile records is a powerful one that should be in the portfolio of every data architect.

There is, however, a downside to profiling records in the data warehouse. Whenever the use of the profile technique is contemplated, note that a certain capability or functionality of the data warehouse is lost. Of necessity, detail is lost whenever aggregation is done. Keep in mind, however, that losing detail is not necessarily a bad thing. The designer of the profile record needs to ensure that the lost detail is not important to the DSS analyst who will ultimately be using the data warehouse. The data architect's first line of defense (and easily the most effective one) is to ensure that such detail is not terribly important is to build the profile records iteratively. By doing so, the data architect has the maneuverability to make changes gracefully. The first iteration of the design of the contents of the profile record suggests the second iteration of design, and so forth. As long as the iterations of data warehouse development are small and fast, there is little danger the end user will find many important requirements left out of the profile record. The danger comes when profile records are created and the first iteration of development is large. In this case, the data architect probably will paint himself or herself into a nasty corner because important detail will have been omitted.

A second approach (which can be used in conjunction with the iterative approach) to ensure that important detail is not permanently lost is to create an alternative level of historical detail along with the profile record, as shown in Figure 3.44. The alternative detail is not designed to be used frequently; it is stored on slow, inexpensive, sequential storage and is difficult to get to and awkward to work with. But the detail is there should it be needed. When management states that they must have a certain detail, however arcane, it can always be retrieved, albeit at a cost of time and money.

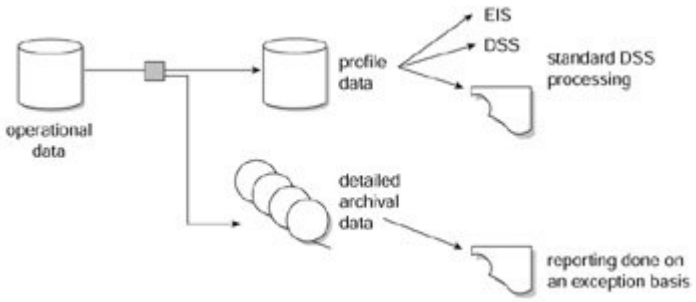

Figure 3.44: An alternative to the classical data warehouse architecture—all the detail that is needed is available while good performance for most DSS processing is the norm.

Creating Multiple Profile Records

Multiple profile records can be created from the same detail. In the case of a phone company, individual call records can be used to create a customer profile record, a district traffic profile record, a line analysis profile record, and so forth.

The profile records can be used to go into the data warehouse or a data mart that is fed by the data warehouse. When the profile records go into a data warehouse, they are for general-purpose use. When the profile records go into the data mart, they are customized for the department that uses the data mart.

The aggregation of the operational records into a profile record is almost always done on the operational server because this server is large enough to manage volumes of data and because that is where the data resides in any case. Usually creating the profile record involves sorting and merging data. Once the process of creating the snapshot becomes complicated and drawn out, whether the snapshot should be taken at all becomes questionable.

The meta data records written for profile records are very similar to the meta data records written for single activity snapshots with the exception that the process of aggregating the records becomes an important piece of meta data. (Technically speaking, the record of the process of aggregation is “meta process” information, not “meta data” information.)

Going from the Data Warehouse to the Operational Environment

The operational environment and the data warehouse environment are about as different as any two environments can be in terms of content, technology, usage, communities served, and a hundred other ways. The interface between the two environments is well documented. Data undergoes a fundamental transformation as it passes from the operational environment to the data warehouse environment. For a variety of reasons—the sequence in which business is conducted, the high performance needs of operational processing, the aging of data, the strong application orientation of operational processing, and so forth—the flow of data from the operational environment to the data warehouse environment is natural and normal. This normal flow of data is shown in Figure 3.45.

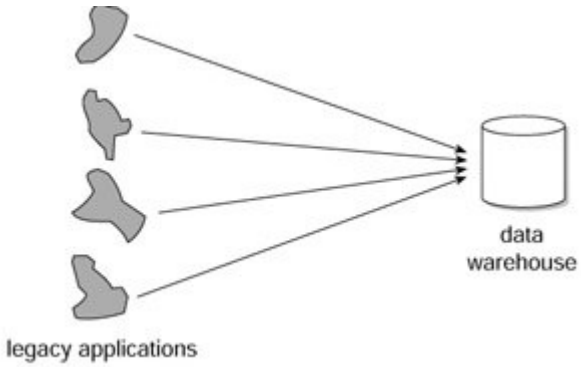

Figure 3.45: The normal flow of data in the legacy application/data warehouse architected environment.

The question occasionally arises, is it possible for data to pass from the data warehouse environment to the operational environment? In other words, can data pass in a reverse direction from that normally experienced? From the standpoint of technology, the answer certainly is yes, such a passage of data is technologically possible. Although it is not normal, there are a few isolated circumstances in which data does indeed flow “backward.”

Direct Access of Data Warehouse Data

Figure 3.46 illustrates the dynamics of the simplest of those circumstances-the direct access of data from the data warehouse by the operational environment. A request has been made within the operational environment for data that resides in the data warehouse. The request is transferred to the data warehouse environment, and the data is located and transferred to

the operational environment. Apparently, from the standpoint of dynamics, the transfer could not be easier.

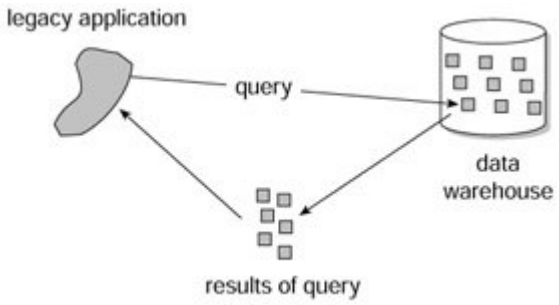

Figure 3.46: A direct query against the data warehouse from the legacy applications environment.

There are a number of serious and uncompromising limitations to the scenario of the direct access of data in the data warehouse. Some of these are as follows:

- The request must be a casual one in terms of response time. It may take as long as 24 hours for it to be satisfied. This means that the operational processing that requires the data warehouse data is decidedly not of an online nature.
- The request for data must be for a minimal amount of data. The data being transferred is measured in terms of bytes, not megabytes or gigabytes.
- The technology managing the data warehouse must be compatible with the technology managing the operational environment in terms of capacity, protocol, and so on.
- The formatting of data after it is retrieved from the data warehouse in preparation for transport to the operational environment must be nonexistent (or minimal).

These conditions preclude most data ever being directly transferred from the data warehouse to the operational environment. It is easy to see why there is a minimal amount of backward flow of data in the case of direct access.

Indirect Access of Data Warehouse Data

Because of the severe and uncompromising conditions of transfer, direct access of data warehouse data by the operational environment is a rare occurrence. Indirect access of data warehouse data is another matter entirely. Indeed, one of the most effective uses of the data warehouse is the indirect access of data warehouse data by the operational environment. Some examples of indirect access of data warehouse data follow.

An Airline Commission Calculation System

One effective indirect use of data warehouse data occurs in the airline environment. Consider, for example, an airline ticketing transaction. A travel agent has contacted the airline reservation clerk on behalf of a customer. The customer has requested a ticket for a flight and the travel agent wants to know the following:

- Is there a seat available?
- What is the cost of the seat?
- What is the commission paid to the travel agent?

If the airline pays too much of a commission, it will get the agent's business, but it will lose money. If the airline pays too little commission, the travel agent will "shop" the ticket and the airline will lose it to another airline that pays a larger commission. It is in the airline's best

interest to calculate the commission it pays very carefully because the calculation has a direct effect on its bottom line.

The interchange between the travel agent and the airline clerk must occur in a fairly short amount of time—within two to three minutes. In this two-to-three-minute window the airline clerk must enter and complete several transactions:

- Are there any seats available?
- Is seating preference available?
- What connecting flights are involved?
- Can the connections be made?
- What is the cost of the ticket?
- What is the commission?

If the response time of the airline clerk (who is running several transactions while carrying on a conversation with the travel agent) starts to be excessive, the airline will find that it is losing business merely because of the poor response time. It is in the best interest of the airline to ensure brisk response time throughout the dialogue with the travel agent.

The calculation of the optimal commission becomes a critical component of the interchange. The optimal commission is best calculated by looking at a combination of two factors—current bookings and the load history of the flight. The current bookings tell how heavily the flight is booked, and the load history yields a perspective of how the flight has been booked in the past. Between current bookings and historical bookings an optimal commission can be calculated.

Though tempting to perform the bookings and flight history calculations online, the amount of data that needs to be manipulated is such that response time suffers if they are calculated in this manner. Instead, the calculation of commission and analysis of flight history are done offline, where there are ample machine resources. Figure 3.47 shows the dynamics of offline commission calculation.

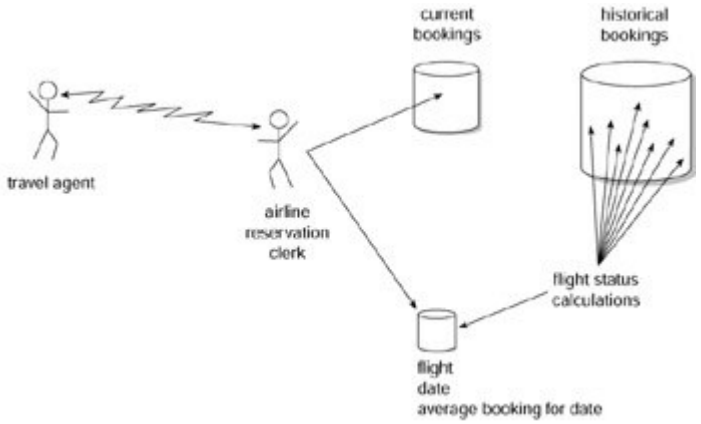

Figure 3.47: The flight status file is created periodically by reading the historical data. It is then a very quick matter for the airline agent to get current bookings and to compare those bookings against the historical average.

The offline calculation and analysis are done periodically, and a small, easy-to-access flight status table is created. When the airline clerk has to interact with the travel agent, it is an easy matter to look at current bookings and the flight status table. The result is a very fast and smooth interaction with the travel agent and the ability to use the data stored in the data warehouse.

A Retail Personalization System

Another example of the indirect use of data warehouse data in the operational environment occurs in the retail personalization system. In this system, a customer reads a catalog or other flyer issued by the retailer. The customer is inspired to make a purchase or to at least inquire about the catalog. A phone call to the retailer ensues.

The interchange is about five to eight minutes long. During this time the retail sales representative has a fair amount of processing to do—identify the customer, take down the specifics of the order, and so forth. The response time is critical; otherwise, the customer will lose interest.

While the customer is placing the order or making an inquiry, the retail sales representative finds out some other information relevant to the interchange, such as the following:

- The last time the customer made a purchase
- The last type of purchase made
- The market segment(s) in which the customer belongs

While engaging the customer in conversation, the sales representative says such things as these:

- "I see it's been since February that we last heard from you."
- "How was that blue sweater you purchased?"
- "Did the problems you had with the pants get resolved?"

In short, the retail sales representative is able to personalize the conversation. The personalization makes the customer more amenable to purchases.

In addition, the retail sales clerk has market segment information available, such as the following:

- Male/female
- Professional/other
- City/country
- Children
- Ages
- Sex
- Sports
- Fishing
- Hunting
- Beach

Because the phone call can be personalized and the direct marketing segment for a customer is available when the customer calls in, the retail sales representative is able to ask pointed questions, such as these:

- "Did you know we have an unannounced sale on swimsuits?"
- "We just got in some Italian sunglasses that I think you might like."
- "The forecasters predict a cold winter for duck hunters. We have a special on waders right now."

The customer has already taken the time to make a phone call. The personalization of the phone call and the knowledge of what products the customer is interested in give the retailer a very good chance at raising revenue with no further outlay of cash or advertising. The personalization of the phone call is achieved by the indirect use of the data warehouse. Figure 3.48 shows the dynamics of how personalization is achieved.

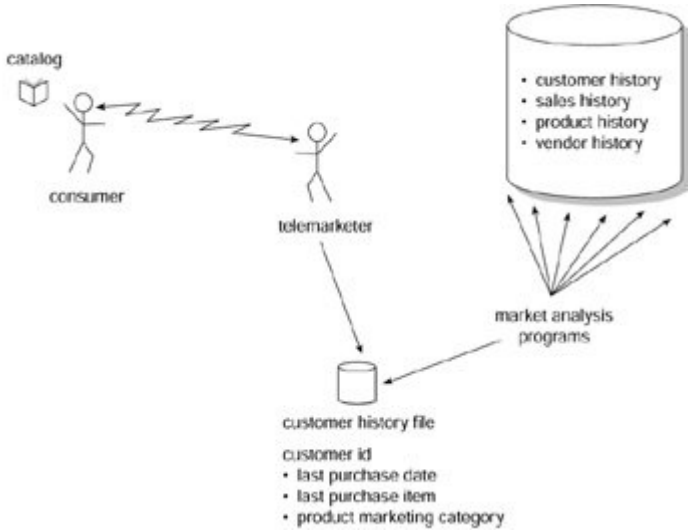

Figure 3.48: The customer history is at the disposal of the telemarketer at a moment's notice.

In the background (i.e., in the data warehouse environment), an analysis program is constantly reading and analyzing customer records. This program scans and analyzes historical customer data in a very sophisticated manner. Periodically, the analysis program spins off a file to the operational environment that contains such information as the following:

- Last purchase date
- Last purchase type
- Market analysis/segmenting

When the customer rings in, the online prepared file is waiting for use by the retail sales representative.

Credit Scoring

Another example of the indirect use of a data warehouse in the operational environment is credit scoring in the banking/finance environment. Credit scoring refers to qualifying (or not qualifying) a person for a loan. Say, for example, a customer walks up to the teller's window and asks for a loan. The teller takes

in some basic information about the customer and decides whether the loan should be approved. The interchange occurs in a very short amount of time—5 to 10 minutes.

To determine whether the loan should be approved, a certain amount of processing must be performed. The loan request is first put through a simple screening process. If the loan is for a small enough amount and if the person has a stable financial background, then it may be approved with no further processing. If the loan is for a fair amount and/or the customer does not have a stable, predictable background, then a further check is required.

The background check relies on the data warehouse. In truth, the check is an eclectic one, in which many aspects of the customer are investigated, such as the following:

- Past payback history
- Home/property ownership
- Financial management
- Net worth
- Gross income
- Gross expenses

- Other intangibles

This extensive background check requires quite a bit of diverse historical data. Completing this part of the loan qualification process requires more than a few minutes.

To satisfy the most customers in the shortest amount of time, an analysis program is written. Figure 3.49 shows how the analysis program fits in with the other components of credit scoring. The analysis program is run periodically and produces a prequalified file for use in the operational environment. In addition to other data, the prequalified file includes the following:

- Customer identification
- Approved credit limit
- Special approval limit

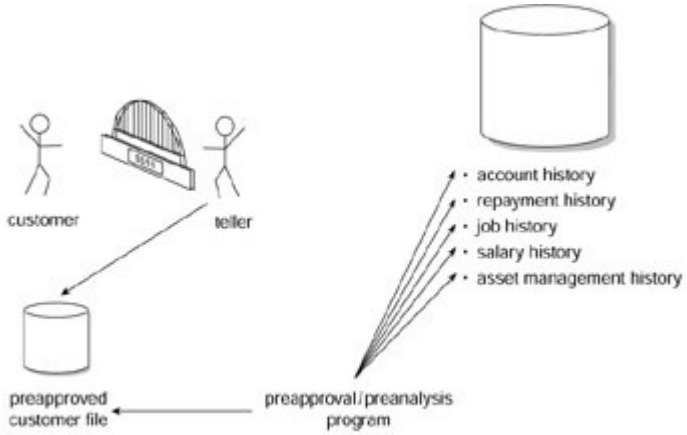

Figure 3.49: The preapproved customer credit file is accessible by the bank teller in an instant.

Now when the customer wishes to apply for and get a loan, in a high-performance, online mode the teller qualifies (or does not qualify) the loan request from the customer. Only if the customer asks for a loan for an amount greater than the preapproved limit does there need to be an interaction by a loan officer.

Indirect Use of Data Warehouse Data

There is, then, an emerging pattern to the indirect use of data warehouse data. That pattern is shown in Figure 3.50.

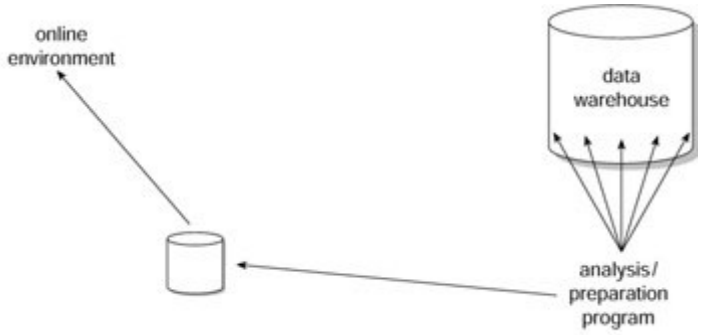

Figure 3.50: The way that data warehouse data is made available to the online operational environment—indirectly.

The data warehouse is analyzed periodically by a program that examines relevant characteristics and criteria. The analysis then creates a small file in the online environment that contains succinct information about the business of the enterprise. The small online file is used quickly and efficiently, fitting in with the style of the other processing that occurs in the operational environment.

Following are a few considerations of the elements of the indirect use of data warehouse data:

- The analysis program:
  - Has many characteristics of artificial intelligence
  - Has free rein to run on any data warehouse data that is available
  - Is run in the background, where processing time is not an issue (or at least not a large issue)
  - Is run in harmony with the rate at which data warehouse changes
- The periodic refreshment:
  - Occurs infrequently
  - Operates in a replacement mode
  - Moves the data from the technology supporting the data warehouse to the technology supporting the operational environment
- The online preanalyzed data file:
  - Contains only a small amount of data per unit of data
  - May contain collectively a large amount of data (because there may be many units of data)
  - Contains precisely what the online clerk needs
  - Is not updated, but is periodically refreshed on a wholesale basis
  - Is part of the online high-performance environment
  - Is efficient to access
  - Is geared for access of individual units of data, not massive sweeps of data

Star Joins

Data warehouse design is decidedly a world in which a normalized approach is the proper one. There are several very good reasons why normalization produces the optimal design for a data warehouse:

- It produces flexibility.
- It fits well with very granular data.
- It is not optimized for any given set of processing requirements.
- It fits very nicely with the data model.

Of course, some small accommodations can be made away from the normalized model when the entire organization views the data in the same way. For example, suppose that monthly data is kept and when the organization looks at the monthly data, it looks at all monthly data. In this case, storing all months together makes sense.

A different approach to database design sometimes mentioned in the context of data warehousing is the multidimensional approach. This approach entails star joins, fact tables, and dimensions. The multidimensional approach applies exclusively to data marts, not data warehouses. Unlike data warehouses, data marts are very much shaped by requirements. To build a data mart, you have to know a lot about the processing requirements that surround the data mart. Once those requirements are known, the data mart can be shaped into an optimal star join structure. But data warehouses are essentially different because they serve a very large community, and as such, they are not optimized for the convenience or performance of any one set of requirements. Data warehouses are shaped around the corporate requirements for information, not the departmental requirements for information. Therefore, creating a star join for the data warehouse is a mistake because the end result will be a data warehouse optimized for one community at the expense of all other communities.

The appeal of the multidimensional approach to database design for data marts begins with the data model. For all of the practical use of a data model as one of the foundations of design, there are some shortcomings. Consider the simple data model in Figure 3.51.

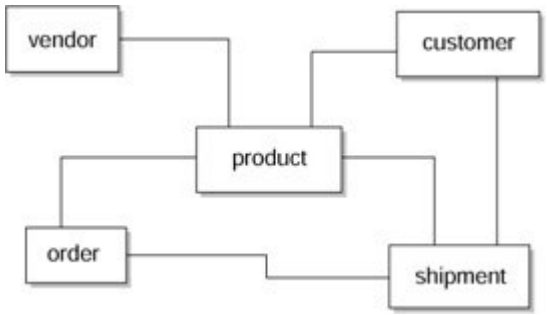

Figure 3.51: A simple two-dimensional data model gives the impression that all entities are equal.

The data model in the figure shows four simple entities with relationships. If all that is considered is the data model for database design, the inference can be drawn that all entities are equal. In other words, from a design standpoint the data model appears to make all entities peers with each other. Approaching database design for the data warehouse solely from the perspective of the data model produces a “flat” effect. In actuality, for a variety of reasons, entities in the world of data marts are anything but peers. Some entities demand their own special treatment.

To see why the data model perspective of the data and the relationships in an organization are distorted, consider the three-dimensional perspective shown in Figure 3.52. Here entities representing vendor, customer, product, and shipment will be sparsely populated, while entities for orders will be heavily populated. There will be many more occurrences of data residing in the table or tables representing the order entity than there will be for any other entity.

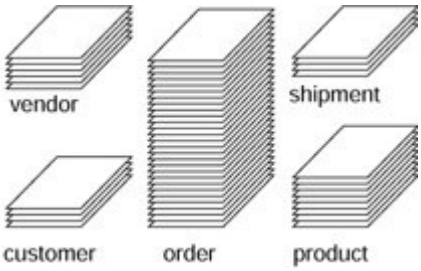

Figure 3.52: A three-dimensional perspective of the entities shows that the entities are anything but equals. Some contain far more occurrences of the data than others.

Because of the massive volume of data populating entity order, a different design treatment is required.

The design structure that is required to manage large amounts of data residing in an entity in a data mart is called a "star join." As a simple example of a star join, consider the data structure shown in Figure 3.53. ORDER is at the center of the star join and is the entity that will be heavily populated. Surrounding ORDER are the entities PART, DATE, SUPPLIER, and SHIPMENT. Each of the surrounding entities will have only a modest number of occurrences of data. The center of the star join-ORDER-is called the "fact table." The surrounding entities—PART, DATE, SUPPLIER, and SHIPMENT—are called "dimension tables." The fact table contains unique identifying data for ORDER, as well as data unique to the order itself. The fact table also contains prejoined foreign key references to tables outlying itself-the dimension tables. The foreign key relationships may be accompanied by nonforeign key information inside the star join if, in fact, the nonforeign key information is used frequently with the fact table. As an example, the description of a PART may be stored inside the fact table along with the PART number if, in fact, the description is used frequently as part of ORDER processing.

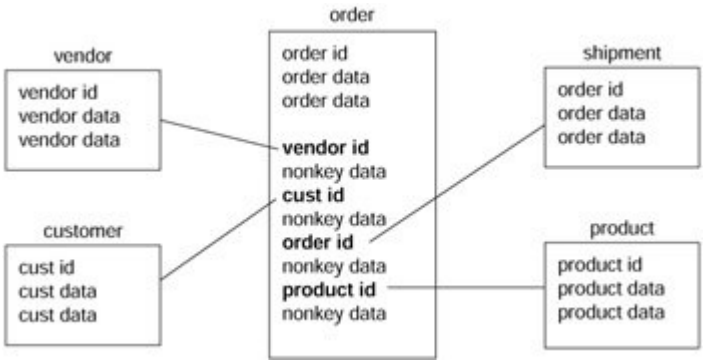

Figure 3.53: A simple star join in which the entity ORDER is populated with many occurrences and other entities are prejoined with the data.

There can be any number of foreign key relationships to the dimension tables. A foreign key relationship is created when there is a need to examine the foreign key data along with data in the fact table.

One of the interesting aspects of the star join is that in many cases textual data is divided from numeric data. Consider the diagram in Figure 3.54. Textual data often ends up in the dimension tables, and numeric data ends up in the fact table. Such a division occurs in almost every case.

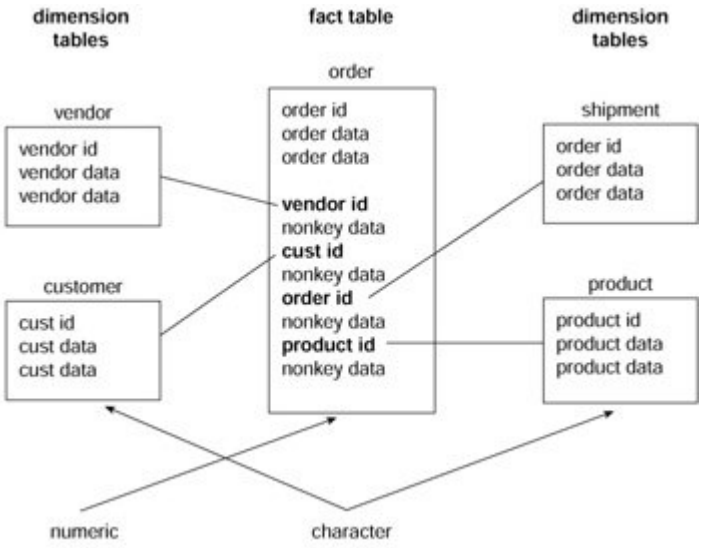

Figure 3.54: In many cases, the fact table is populated by numeric data and foreign keys, while the dimension table is populated by character data.

The benefit of creating star joins is to streamline data for DSS processing. By prejoining data and by creating selective redundancy, the designer greatly simplifies and streamlines data for access and analysis, which is exactly what is needed for the data mart. Note that if star joins were used outside of the DSS data mart environment, there would be many drawbacks. Outside the DSS data mart environment, where update occurs and where data relationships are managed up to the second, a star join most likely would be a very cumbersome structure to build and maintain. But because the data mart is a load-and-access environment, because the data mart contains historical data, and because massive amounts of data need to be managed, the star join data structure is ideal for the processing that occurs inside the star join.

The star join then has its rightful place as a foundation for data mart design. Figure 3.55 shows how the star join and the data model fit as foundations for data mart DSS design. The star join applies as a design foundation to the very large entities that will exist in the data mart. The data model applies as a design foundation to the nonvoluminous entities found in the data mart.

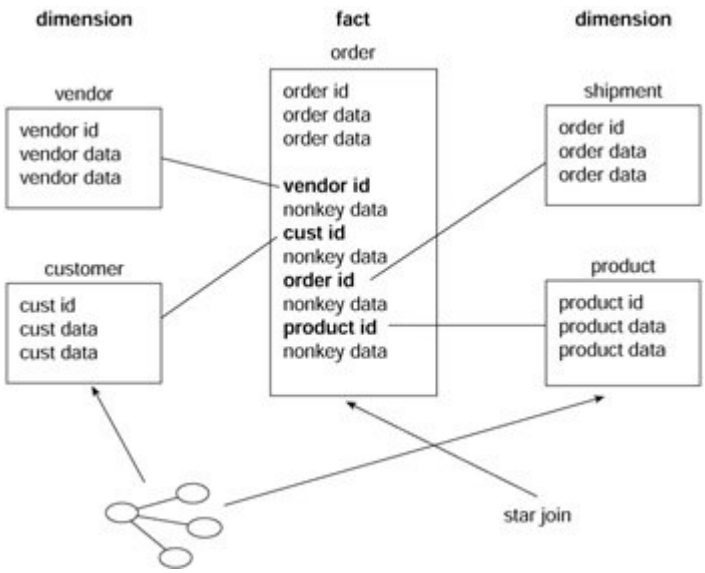

Figure 3.55: Classical data modeling applies to the dimension tables (i.e., the nonpopulous entities) and star join design applies to the fact tables (i.e., the populous entities).

One of the issues of data warehouses and data marts is how data gets from the data warehouse to the data mart. Data in the data warehouse is very granular. Data in the data mart is very compact and summarized. Periodically data must be moved from the data warehouse to the data mart. This movement of data from the data warehouse to the data mart is analogous to the movement of data into the data warehouse from the operational legacy environment.

Data Marts: A Substitute for a Data Warehouse?

There is an argument in the IT community that says that a data warehouse is expensive and troublesome to build. Indeed, a data warehouse requires resources in the best of cases. But building a data warehouse is absolutely worth the effort. The argument for not building a data warehouse usually leads to building something short of a data warehouse, usually a data mart. The premise is that you can get a lot out of a data mart without the high cost and investment for a data warehouse.

From a short-term perspective, there is some merit to this argument. But from a long-term perspective, a data mart is never a substitute for a data warehouse. Figure 3.56 shows why.

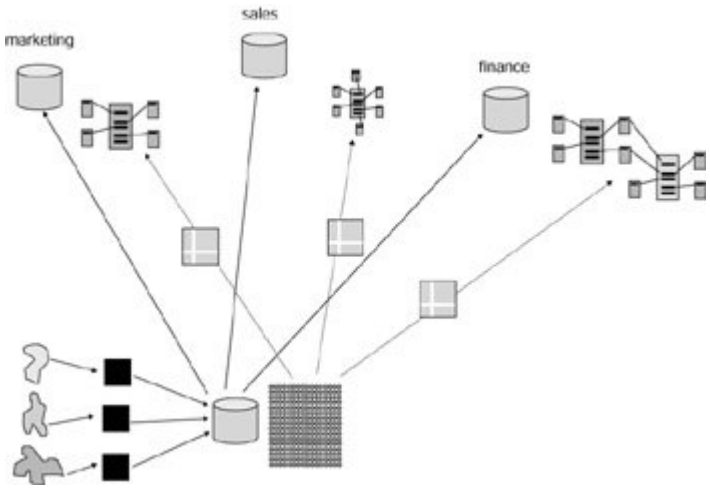

Figure 3.56: The relationship between the data warehouse and the data mart.

The structure of the data found in the data mart is shaped by the particular requirements of the department. The finance department will have one structure for its data mart, the sales department will have another structure for its data mart, and the marketing department will have another data structure for its data mart. All of their structures will be fed from the granular data found in the data warehouse.

The data structure found in any given data mart is different from the data structure for any other data mart. For example, the sales data mart data structure will be different from the marketing data mart data. The data mart structures are typically known as star joins and contain fact tables and dimensions. The data mart structures are typically known as multidimensional structures and are served by OLAP technology.

Because there is a different data structure for each data mart, making any data mart into a data warehouse doesn't make sense. When a data mart star join is made into a data warehouse, the data warehouse is optimal for one data mart and its users and is not optimal (or really usable) for anyone else. Data marts produce structures that are not reusable except by someone operating in the department that is optimized.

Data mart data structures—in general, across the enterprise—are not reusable, are not flexible, are not useful as a foundation for reconciliation, and are not standing ready for a new set of unknown requirements. But the normalized granular data found in a data warehouse is indeed all of those things.

Data warehouse data must be selected, accessed, and then reshaped to meet the needs of the data mart. Often the data mart data resides in cubes. The cubes need to be formed, and many different calculations need to be performed on the detailed data that resides in the data warehouse. In short, a nontrivial process occurs as data is passed from a normalized world into a multidimensional world.

One of the important issues here is how much data must be accessed and how often is the refreshment process to be performed.

Supporting the ODS

In general there are three classes of ODS—class I, class II, and class III. In a class I ODS, updates of data from the operational environment to the ODS are synchronous. In a class II ODS, the updates between the operational environment and the ODS occur within a two-to-

three-hour time frame. And in a type III ODS, the synchronization of updates between the operational environment and the ODS occurs overnight. But there is another type of ODS structure—a class IV ODS, in which updates into the ODS from the data warehouse are unscheduled. Figure 3.57 shows this support.

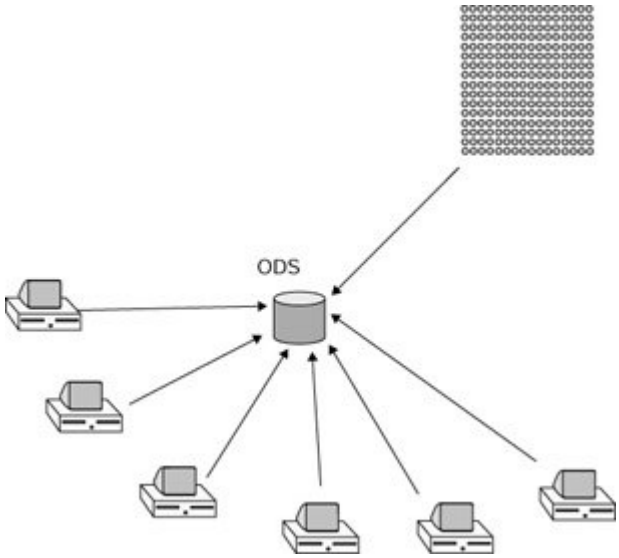

Figure 3.57: The data warehouse supports a class IV ODS.

The data in the data warehouse is analyzed, and periodically the data is placed in the ODS. The data that is shipped to the ODS is shipped in the form of profile data, which is data that represents many different physical occurrences of data. As a simple example of profile data, suppose the details of a customer's transactions are analyzed. The customer has been active for several years. The analysis of the transactions in the data warehouse is used to produce the following profile information about a single customer:

- Customer name and ID
- Customer volume—high/low
- Customer profitability—high/low
- Customer frequency of activity—very frequent/very infrequent
- Customer likes/dislikes (fast cars, beautiful women, single malt scotch)

Each of the categories of information found in the profile record is created from the examination and analysis of the many detailed records found in the data warehouse. There is then a very fundamental difference between the data found in the data warehouse and the profile data found in the class IV ODS.

Summary

The design of the data warehouse begins with the data model. The corporate data model is used for the design of the operational environment, and a variation of the corporate data model is used for the data warehouse. The data warehouse is constructed in an iterative fashion. Requirements for the data warehouse cannot be known a priori. The construction of the data warehouse is under a development life cycle completely different from that of classical operational systems.

The primary concern of the data warehouse developer is managing volume. To that end, granularity and partitioning of data are the two most important issues of database design.

There are, however, many other physical design issues, most of which center around the efficiency of access to data.

The data warehouse is fed data as it passes from the legacy operational environment. Data goes through a complex process of conversion, reformatting, and integration as it passes from the legacy operational environment into the data warehouse environment. Often, as data passes into the data warehouse environment there is a shift of time. In some cases, the operational data has no timestamping, and in other cases, the level of granularity of the operational data needs to be adjusted.

The data model exists at three levels—high level, midlevel, and low level. The data model is the key to being able to build the data warehouse in iterations. The entities found in the high-level model relate to the major subject areas of the corporation. The low-level model relates to the physical database design of the data warehouse.

At the lowest level of database design, slight denormalization can occur when the entire organization looks at the data in the same way. Some techniques for slight denormalization of data include creating arrays of data, creating redundant data judiciously, and making creative indexes.

The basic structure of the data warehouse record is one that contains a timestamp, a key, direct data, and secondary data. All data warehouse database designs—in one form or the other—follow this simple pattern.

Reference tables need to be placed in the data warehouse and managed on a time-variant basis just like any other data. There are many approaches to the inclusion and design of reference data in the data warehouse.

Data is loaded into the data warehouse under what can be termed a “wrinkle of time.” This means that as soon as an activity occurs in the operational environment, that data is not immediately rushed to the data warehouse. Instead, data that has been newly updated in the operational environment is allowed to stand in the operational environment for up to 24 hours before being moved to the data warehouse.

The transformation that occurs as data moves from the operational environment to the data warehouse environment is complex. There is a change in DBMS, a change in operating systems, a change in hardware architecture, a change in semantics, a change in coding, and so forth. Many, many considerations are involved in the movement of data from the operational environment to the data warehouse environment.

The creation of a data warehouse record is triggered by an activity or an event that has occurred in the operational environment. In some cases, an event—such as a sale—has occurred. In other cases, the regular passage of time has occurred, such as the end of the month or the end of the week.

A profile record is a composite record made up of many different historical activities. The profile record is a composite representation of data.

The star join is a database design technique that is sometimes mistakenly applied to the data warehouse environment. The star join multidimensional approach is an approach where database design is based on the occurrences of data within a subject area and how that data will be accessed. Star join design applies to the world of data marts, not the world of data warehouses. It is a mistake to build a data warehouse with a star join because the data warehouse will end up being optimal for one set of users at the expense of everyone else.

Chapter 4: Granularity in the Data Warehouse

Overview

The single most important design issue facing the data warehouse developer is determining the granularity. When the granularity is properly set, the remaining aspects of design and implementation flow smoothly; when it is not properly set, every other aspect is awkward.

Granularity is also important to the warehouse architect because it affects all of the environments that depend on the warehouse for data. Granularity affects how efficiently data can be shipped to the different environments determines the types of analysis that can be done.

The primary issue of granularity is that of getting it at the right level. The level of granularity needs to be neither too high or too low.

The trade-off in choosing the right levels of granularity—as discussed in Chapter 2—centers around managing the volume of data and storing data at too high a level of granularity, to the point that detailed data is so voluminous that it is unusable. In addition, if there is to be a truly large amount of data, consideration must be given to putting the inactive portion of the data into overflow storage.

Raw Estimates

The starting point for determining the appropriate level of granularity is to do a raw estimate of the number of rows of data and the DASD (direct access storage device) that will be in the data warehouse. Admittedly, in the best of circumstances, only an estimate can be made. But all that is required at the inception of building the warehouse is an order-of-magnitude estimate.

The raw estimate of the number of rows of data that will reside in the data warehouse tells the architect a great deal. If there are only 10,000 rows, almost any level of granularity will do. If there are 10 million rows, a low level of granularity is needed. If there are 10 billion rows, not only is a low level of granularity needed, but a major portion of the data must go into overflow storage.

Figure 4.1 shows an algorithmic path to calculate the space occupied by a data warehouse. The first step is to identify all the tables to be built. As a rule of thumb, there will be one or two really large tables and many smaller supporting tables. Next, estimate the size of the row in each table. It is likely that the exact size will not be known. A lower-bound estimate and an upper-bound estimate are sufficient.

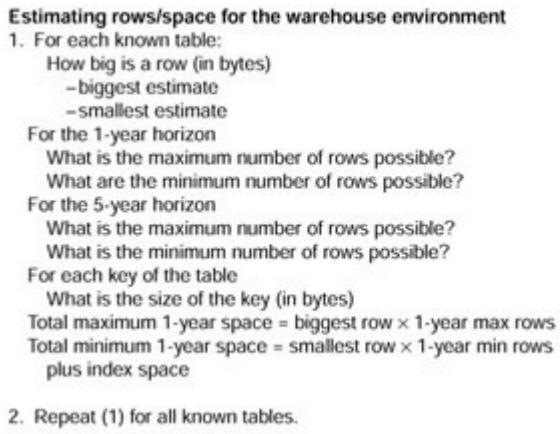

Figure 4.1: Space/row calculations.

Next, on the one-year horizon, estimate the maximum and minimum number of rows in the table. This estimate is the one that is the most problematic for the designer. If the table is for customers, use today's estimate, factoring in business conditions and the corporate business plan. If there is no existing business today, estimate the total market multiplied by the expected market share. If the market share is unpredictable, use an estimate of what a competitor has achieved. In short, start with a reasonable estimate of customers gathered from one or more sources.

If the warehouse is to contain information about activity, go from the estimated number of customers to the estimated activity per unit of time. Again, the same logic is used: looking at current business profiles, a competitor's profile, economic projections, and so forth.

Once the estimate of number of units of data in the data warehouse is made (using a high and a low projection), repeat the process, but this time for the five-year horizon.

After the raw data projections are made, the index data space projections are calculated. For each table—for each key in the table or element of data that will be searched directly—identify the length of the key or element of data and determine whether the key will exist for each entry in the primary table.

Now the high and low numbers for the occurrences of rows in the tables are multiplied, respectively, by the maximum and minimum lengths of data. In addition, the number of index entries is multiplied by the length of the key and added to the total amount of data in order to determine the volume of data that will be required.

A word of caution: Estimates projecting the size of the data warehouse almost always are low. Furthermore, the growth rate of the warehouse is usually faster than the projection.

Input to the Planning Process

The estimate of rows and DASD then serves as input to the planning process, as shown by Figure 4.2. When the estimates are made, accuracy is actually important (or even desirable) only to the order of magnitude. A fine degree of accuracy here is a waste of time.

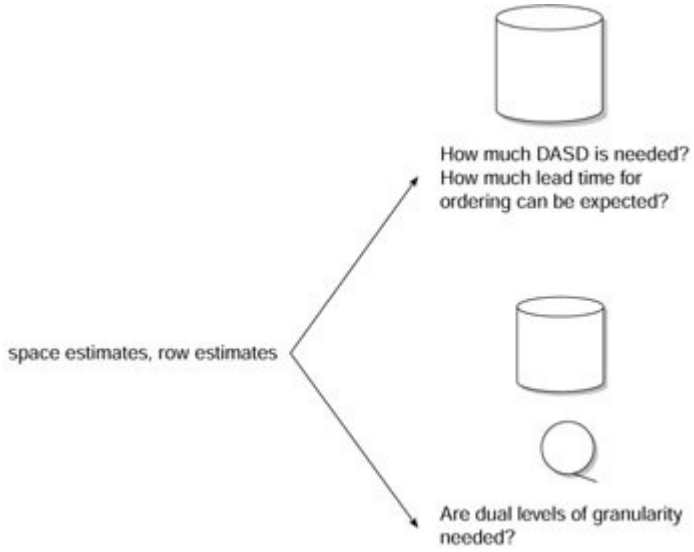

Figure 4.2: Using the output of the space estimates.

Data in Overflow?

Once the raw estimate as to the size of the data warehouse is made, the next step is to compare the total number of rows in the warehouse environment to the charts shown in Figure 4.3. Depending on how many total rows will be in the warehouse environment, different approaches to design, development, and storage are necessary. For the one-year horizon, if the number of row total fewer than 100,000, practically any design and implementation will work, and no data will have to go to overflow. If there will be 1 million total rows or fewer, design must be done carefully, and it is unlikely that any data will have to go into overflow. If the total number of row will exceed 10 million, design must be done carefully, and it is likely that at least some data will go to overflow. And if the total number of rows in the data warehouse environment is to exceed 100 million rows, surely a large amount of data will go to overflow storage, and a very careful design and implementation of the data warehouse is required.

1-year horizon		5-year horizon	
100,000,000	data in overflow and on disk, majority in overflow, very careful consideration of granularity	1,000,000,000	data in overflow and on disk, majority in overflow, very careful consideration of granularity
10,000,000	possibly some data in overflow, most data on disk, some consideration of granularity	100,000,000	possibly some data in overflow, most data on disk, some consideration of granularity
1,000,000	data on disk, almost any database design	10,000,000	data on disk, almost any database design
100,000	any database design, all data on disk	1,000,000	any database design, all data on disk

Figure 4.3: Compare the total number of rows in the warehouse environment to the charts.

On the five-year horizon, the totals shift by about an order of magnitude. The theory is that after five years these factors will be in place:

- There will be more expertise available in managing the data warehouse volumes of data.
- Hardware costs will have dropped to some extent.
- More powerful software tools will be available.
- The end user will be more sophisticated.

All of these factors point to a different volume of data that can be managed over a long period of time. Unfortunately, it is almost impossible to accurately forecast the volume of data into a five-year horizon. Therefore, this estimate is used as merely a raw guess.

An interesting point is that the total number of bytes used in the warehouse has relatively little to do with the design and granularity of the data warehouse. In other words, it does not particularly matter whether the record being considered is 25 bytes long or 250 bytes long. As long as the length of the record is of reasonable size, then the chart shown in Figure 4.3 still applies. Of course, if the record being considered is 250,000 bytes long, then the length of the record makes a difference. Not many records of that size are found in the data warehouse environment, however. The reason for the indifference to record size has as much to do with the indexing of data as anything else. The same number of index entries is required regardless of the size of the record being indexed. Only under exceptional circumstances does the actual size of the record being indexed play a role in determining whether the data warehouse should go into overflow.

Overflow Storage

Data in the data warehouse environment grows at a rate never before seen by IT professionals. The combination of historical data and detailed data produces a growth rate that is phenomenal. The terms terabyte and petabyte were used only in theory prior to data warehousing.

As data grows large a natural subdivision of data occurs between actively used data and inactively used data. Inactive data is sometimes called *dormant data*. At some point in the life of the data warehouse, the vast majority of the data in the warehouse becomes stale and unused. At this point it makes sense to start separating the data onto different storage media.

Most professionals have never built a system on anything but disk storage. But as the data warehouse grows large, it simply makes economic and technological sense to place the data on multiple storage media. The actively used portion of the data warehouse remains on disk storage, while the inactive portion of the data in the data warehouse is placed on alternative storage or near-line storage.

Data that is placed on alternative or near-line storage is stored much less expensively than data that resides on disk storage. And just because data is placed on alternative or near-line storage does not mean that the data is inaccessible. Data placed on alternate or near-line storage is just as accessible as data placed on disk storage. By placing inactive data on alternate or near-line storage, the architect removes impediments to performance from the high-performance active data. In fact, moving data to near-line storage greatly accelerates the performance of the entire environment.

To make data accessible throughout the system and to place the proper data in the proper part of storage, software support of the alternate storage/near-line environment is needed. Figure 4.4 shows some of the more important components of the support infrastructure needed for the alternate storage/near-line storage environment.

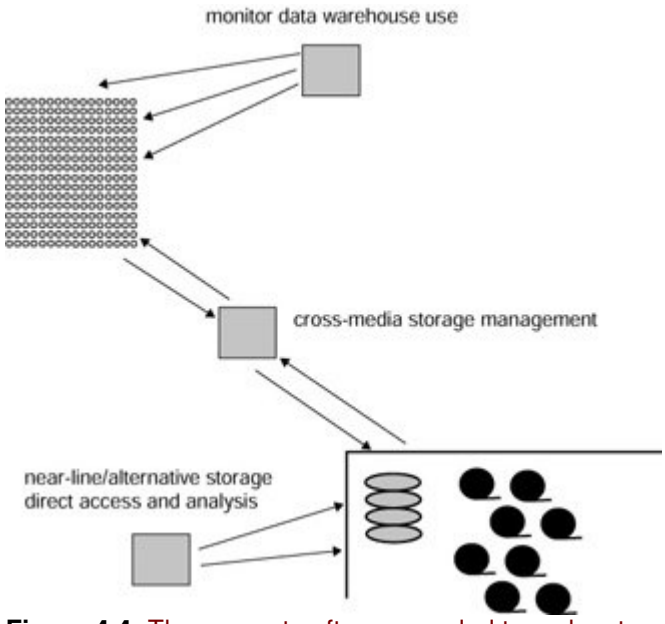

Figure 4.4: The support software needed to make storage overflow possible.

Figure 4.4 shows that a data monitor is needed to determine the usage of data. The data monitor tells where to place data. The movement between disk storage and near-line storage is controlled by means of software called a cross-media storage manager. The data in alternate storage/near-line storage can be accessed directly by means of software that has the intelligence to know where data is located in near-line storage. These three software components are the minimum required for alternate storage/near-line storage to be used effectively.

In many regards alternate storage/near-line storage acts as overflow storage for the data warehouse. Logically, the data warehouse extends over both disk storage and alternate storage/near-line storage in order to form a single image of data. Of course, physically the data may be placed on any number of volumes of data.

An important component of the data warehouse is overflow storage, where infrequently used data is held. Overflow storage has an important effect on granularity. Without this type of storage, the designer is forced to adjust the level of granularity to the capacity and budget for disk technology. With overflow storage the designer is free to create as low a level of granularity as desired.

Overflow storage can be on any number of storage media. Some of the popular media are photo optical storage, magnetic tape (sometimes called “near-line storage”), and cheap disk. The magnetic tape storage medium is not the same as the old-style mag tapes with vacuum units tended by an operator. Instead, the modern rendition is a robotically controlled silo of storage where the human hand never touches the storage unit.

The alternate forms of storage are cheap, reliable, and capable of storing huge amounts of data, much more so than is feasible for storage on high-performance disk devices—the alternate of storage. In doing so, the alternate forms of storage as overflow for the data warehouse allow. In some cases, a query facility that can operate independently of the storage device is desirable. In this case when a user makes a query there is no prior knowledge of where the data resides. The query is issued, and the system then finds the data regardless of where it is.

While it is convenient for the end user to merely “go get the data,” there is a performance implication. If the end user frequently accesses data that is in alternate storage, the query will not run quickly, and many machine resources will be consumed in the servicing of the request. Therefore, the data architect is best advised to make sure that the data that resides in alternate storage is accessed infrequently.

There are several ways to ensure infrequently accessed data resides in alternate storage. A simple way is to place data in alternate storage when it reaches a certain age—say, 24 months. Another way is to place certain types of data in alternate storage and other types in disk storage. Monthly summary of customer records may be placed in disk storage, while details that support the monthly summary are placed in alternate storage.

In other cases of query processing, separating the disk-based queries from the alternate-storage-based queries is desirable. Here, one type of query goes against disk-based storage and another type goes against alternate storage. In this case, there is no need to worry about the performance implications of a query having to fetch alternate-storage-based data.

This sort of query separation can be advantageous—particularly with regard to protecting systems resources. Usually the types of queries that operate against alternate storage end up accessing huge amounts of data. Because these long-running activities are performed in a completely separate environment, the data administrator never has to worry about query performance in the disk-based environment.

For the overflow storage environment to operate properly, several types of software become mandatory. Figure 4.5 shows these types and where they are positioned.

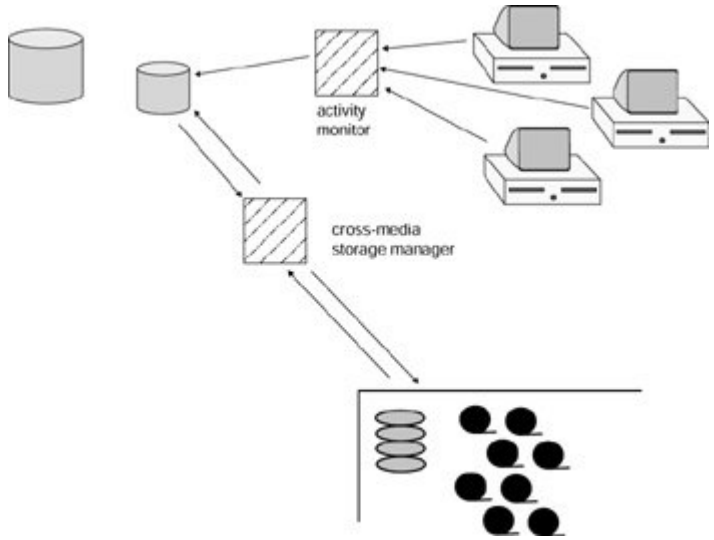

Figure 4.5: For overflow storage to function properly, at least two types of software are needed—a cross-media storage manager and an activity monitor.

Figure 4.5 shows that two pieces of software are needed for the overflow environment to operate properly—a cross-media storage manager and an activity monitor. The cross-media storage manager manages the traffic of data going to and from the disk storage environment to the alternate storage environment. Data moves from the disk to alternate storage when it ages or when its probability of access drops. Data from the alternate storage environment can be moved to disk storage when there is a request for the data or when it is detected that there will be multiple future requests for the data. By moving the data to and from disk storage to alternate storage, the data administrator is able to get maximum performance from the system.

The second piece required, the activity monitor, determines what data is and is not being accessed. The activity monitor supplies the intelligence to determine where data is to be placed—on disk storage or on alternate storage.

What the Levels of Granularity Will Be

Once the simple analysis is done (and, in truth, many companies discover that they need to put at least some data into overflow storage), the next step is to determine the level of granularity for data residing on disk storage. This step requires common sense and a certain amount of intuition. Creating a disk-based data warehouse at a very low level of detail doesn't make sense because too many resources are required to process the data. On the other hand, creating a disk-based data warehouse with a level of granularity that is too high means that much analysis must be done against data that resides in overflow storage. So the first cut at determining the proper level of granularity is to make an educated guess.

Such a guess is only the starting point, however. To refine the guess, a certain amount of iterative analysis is needed, as shown in Figure 4.6. The only real way to determine the proper level of granularity for the lightly summarized data is to put the data in front of the end user. Only after the end user has actually seen the data can a definitive answer be given. Figure 4.6 shows the iterative loop that must transpire.

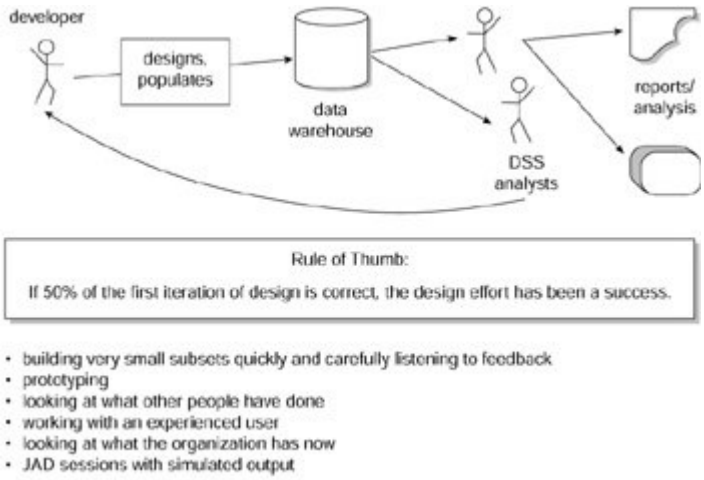

Figure 4.6: The attitude of the end user: "Now that I see what can be done, I can tell you what would really be useful."

The second consideration in determining the granularity level is to anticipate the needs of the different architectural entities that will be fed from the data warehouse. In some cases, this determination can be done scientifically. But, in truth, this anticipation is really an educated guess. As a rule, if the level of granularity in the data warehouse is small enough, the design of the data warehouse will suit all architectural entities. Data that is too fine can always be summarized, whereas data that is not fine enough cannot be easily broken down. Therefore, the data in the data warehouse needs to be at the lowest common denominator.

Some Feedback Loop Techniques

Following are techniques to make the feedback loop harmonious:

- Build the first parts of the data warehouse in very small, very fast steps, and carefully listen to the end users' comments at the end of each step of development. Be prepared to make adjustments quickly.
- If available, use prototyping and allow the feedback loop to function using observations gleaned from the prototype.
- Look at how other people have built their levels of granularity and learn from their experience.
- Go through the feedback process with an experienced user who is aware of the process occurring. Under no circumstances should you keep your users in the dark as to the dynamics of the feedback loop.
- Look at whatever the organization has now that appears to be working, and use those functional requirements as a guideline.
- Execute joint application design (JAD) sessions and simulate the output in order to achieve the desired feedback.

Granularity of data can be raised in many ways, such as the following:
- Summarize data from the source as it goes into the target.
- Average or otherwise calculate data as it goes into the target.
- Push highest/lowest set values into the target.
- Push only data that is obviously needed into the target.
- Use conditional logic to select only a subset of records to go into the target.

The ways that data may be summarized or aggregated are limitless.

When building a data warehouse, keep one important point in mind. In classical requirements systems development, it is unwise to proceed until the vast majority of the requirements are identified. But in building the data warehouse, it is unwise not to proceed if at least half of the requirements for the data warehouse are identified. In other words, if in building the data warehouse the developer waits until many requirements are identified, the warehouse will never be built. It is vital that the feedback loop with the DSS analyst be initiated as soon as possible.

As a rule, when transactions are created in business they are created from lots of different types of data. An order contains part information, shipping information, pricing, product specification information, and the like. A banking transaction contains customer information, transaction amounts, account information, banking domicile information, and so forth. When normal business transactions are being prepared for placement in the data warehouse, their level of granularity is too high, and they must be broken down into a lower level. The normal circumstance then is for data to be broken down. There are at least two other circumstances in which data is collected at too low a level of granularity for the data warehouse, however:
- Manufacturing process control. Analog data is created as a by-product of the manufacturing process. The analog data is at such a deep level of granularity that it is not useful in the data warehouse. It needs to be edited and aggregated so that its level of granularity is raised.
- Clickstream data generated in the Web environment. Web logs collect clickstream data at a granularity that it is much too fine to be placed in the data warehouse. Clickstream data must be edited, cleansed, resequenced, summarized, and so forth before it can be placed in the warehouse.

These are a few notable exceptions to the rule that business-generated data is at too high a level of granularity.

Levels of Granularity—Banking Environment

Consider the simple data structures shown in Figure 4.7 for a banking/financial environment.

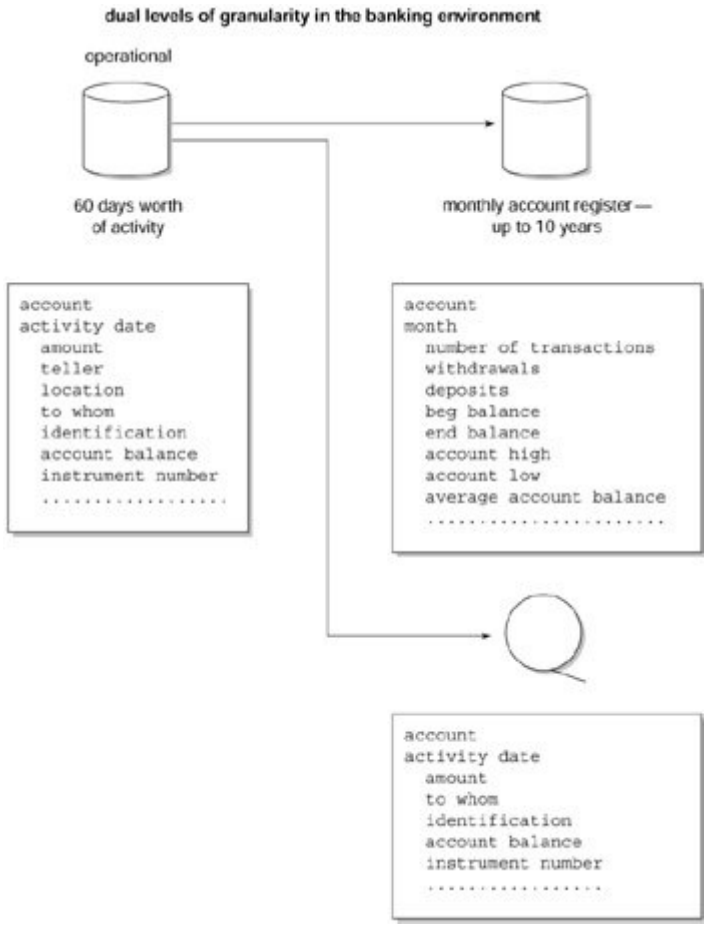

Figure 4.7: A simple example of dual levels of granularity in the banking environment.

To the left—at the operational level—is operational data, where the details of banking transactions are found. Sixty days' worth of activity are stored in the operational online environment.

In the lightly summarized level of processing—shown to the right of the operational data—are up to 10 years' history of activities. The activities for an account for a given month are stored in the lightly summarized portion of the data warehouse. While there are many records here, they are much more compact than the source records. Much less DASD and many fewer rows are found in the lightly summarized level of data.

Of course, there is the archival level of data (i.e., the overflow level of data), in which every detailed record is stored. The archival level of data is stored on a medium suited to bulk management of data. Note that not all fields of data are transported to the archival level. Only those fields needed for legal reasons, informational reasons, and so forth are stored. The data that has no further use, even in an archival mode, is purged from the system as data is passed to the archival level.

The overflow environment can be held in a single medium, such as magnetic tape, which is cheap for storage and expensive for access. It is entirely possible to store a small part of the archival level of data online, when there is a probability that the data might be needed. For example, a bank might store the most recent 30 days of activities online. The last 30 days is archival data, but it is still online. At the end of the 30-day period, the data is sent to magnetic tape, and space is made available for the next 30 days' worth of archival data.

Now consider another example of data in an architected environment in the banking/financial environment. Figure 4.8 shows customer records spread across the environment. In the operational environment is shown current-value data whose content is accurate as of the moment of usage. The data that exists at the light level of summarization is the same data (in terms of definition of data) but is taken as a snapshot once a month.

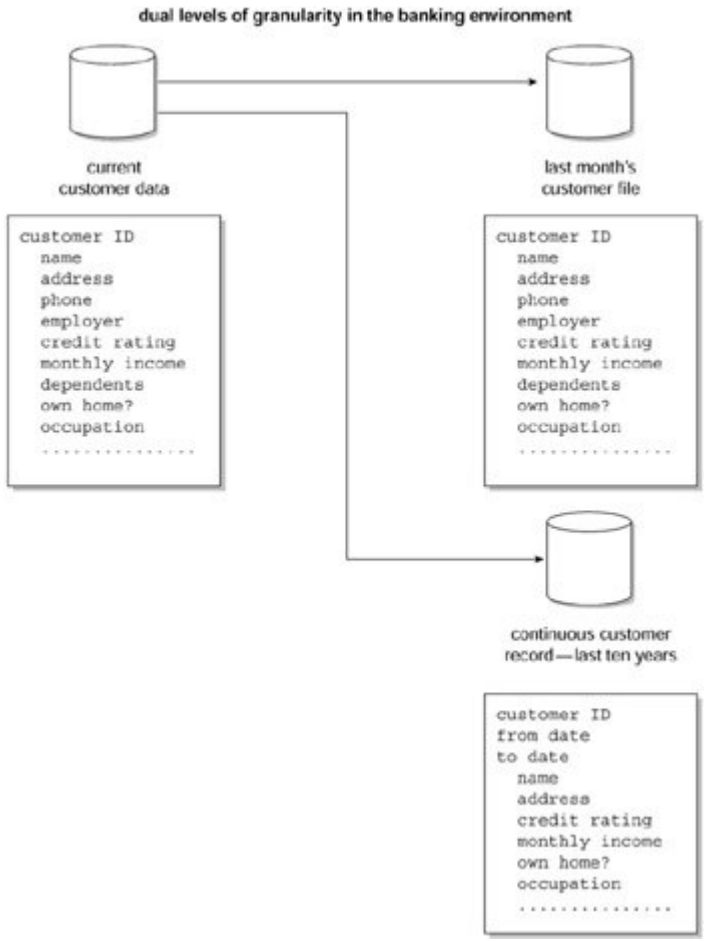

Figure 4.8: Another form of dual levels of granularity in the banking environment.

Where the customer data is kept over a long span of time—for the past 10 years-a continuous file is created from the monthly files. In such a fashion the history of a customer can be tracked over a lengthy period of time.

Now let's move to another industry—manufacturing. In the architected environment shown in Figure 4.9, at the operational level is the record of manufacture upon the completion of an assembly for a given lot of parts. Throughout the day many records aggregate as the assembly process runs.

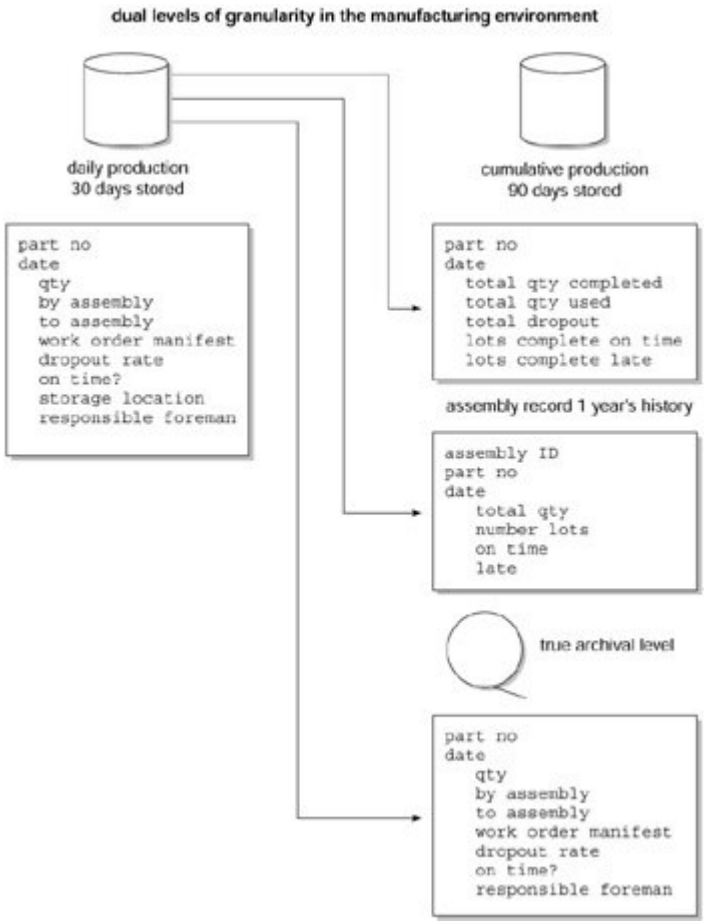

Figure 4.9: Some of the different levels of granularity in a manufacturing environment.

The light level of summarization contains two tables—one for all the activities for a part summarized by day, another by assembly activity by part. The parts' cumulative production table contains data for up to 90 days. The assembly record contains a limited amount of data on the production activity summarized by date.

The archival/overflow environment contains a detailed record of each manufacture activity. As in the case of a bank, only those fields that will be needed later are stored. (Actually, those fields that have a reasonable probability of being needed later are stored.)

Another example of data warehouse granularity in the manufacturing environment is shown in Figure 4.10, where an active-order file is in the operational environment. All orders that require activity are stored there. In the data warehouse is stored up to 10 years' worth of order history. The order history is keyed on the primary key and several secondary keys. Only the data that will be needed for future analysis is stored in the warehouse. The volume of orders was so small that going to an overflow level was not necessary. Of course, should orders suddenly increase, it may be necessary to go to a lower level of granularity and into overflow.

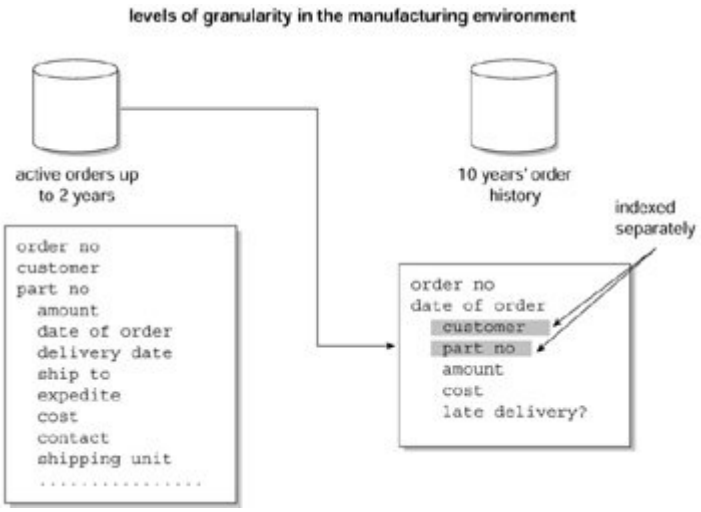

Figure 4.10: There are so few order records that there is no need for a dual level of granularity.

Another adaptation of a shift in granularity is seen in the data in the architected environment of an insurance company, shown in Figure 4.11. Premium payment information is collected in an active file. Then, after a period of time, the information is passed to the data warehouse. Because only a relatively small amount of data exists, overflow data is not needed. However, because of the regularity of premium payments, the payments are stored as part of an array in the warehouse.

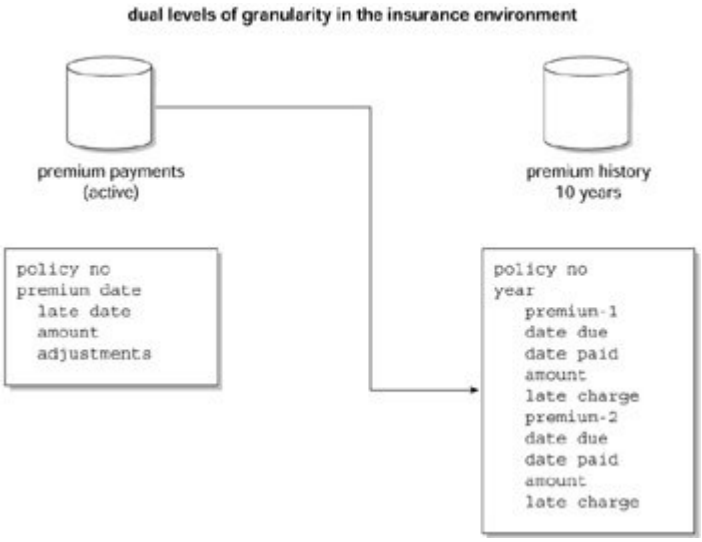

Figure 4.11: Because of the low volume of premiums, there is no need for dual levels of granularity, and because of the regularity of premium billing, there is the opportunity to create an array of data.

As another example of architecture in the insurance environment, consider the insurance claims information shown in Figure 4.12. In the current claims system (the operational part of the environment), much detailed information is stored about claims. When a claim is settled (or when it is determined that a claim is not going to be settled), or when enough time passes that the claim is still pending,

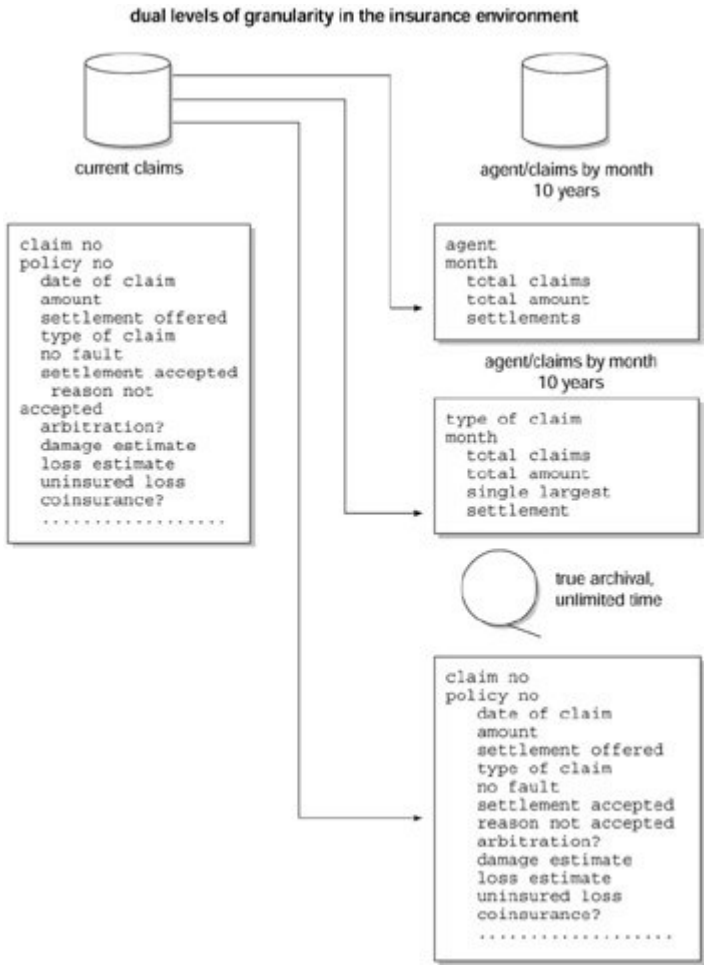

Figure 4.12: Claims information is summarized on other than the primary key in the lightly summarized part of the warehouse. Claims information must be kept indefinitely in the true archival portion of the architecture.

the claim information passes over to the data warehouse. As it does so, the claim information is summarized in several ways—by agent by month, by type of claim by month, and so on. At a lower level of detail, the claim is held in overflow storage for an unlimited amount of time. As in the other cases in which data passes to overflow, only data that might be needed in the future is kept (which is most of the information found in the operational environment).

Summary

Choosing the proper levels of granularity for the architected environment is vital to success. The normal way the levels of granularity are chosen is to use common sense, create a small part of the warehouse, and let the user access the data. Then listen very carefully to the user, take the feedback he or she gives, and adjust the levels of granularity appropriately.

The worst stance that can be taken is to design all the levels of granularity a priori, then build the data warehouse. Even in the best of circumstances, if 50 percent of the design is done correctly, the design is a good one. The nature of the data warehouse environment is such that the DSS analyst cannot envision what is really needed until he or she actually sees the reports.

The process of granularity design begins with a raw estimate of how large the warehouse will be on the one-year and the five-year horizon. Once the raw estimate is made, then the estimate tells the designer just how fine the granularity should be. In addition, the estimate tells whether overflow storage should be considered.

There is an important feedback loop for the data warehouse environment. Upon building the data warehouse's first iteration, the data architect listens very carefully to the feedback from the end user. Adjustments are made based on the user's input.

Another important consideration is the levels of granularity needed by the different architectural components that will be fed from the data warehouse. When data goes into overflow—away from disk storage to a form of alternate storage—the granularity can be as low as desired. When overflow storage is not used, the designer will be constrained in the selection of the level of granularity when there is a significant amount of data.

For overflow storage to operate properly, two pieces of software are necessary—a cross-media storage manager that manages the traffic to and from the disk environment to the alternate storage environment and an activity monitor. The activity monitor is needed to determine what data should be in overflow and what data should be on disk.

Chapter 5: The Data Warehouse and Technology

In many ways, the data warehouse requires a simpler set of technological features than its predecessors. Online updating with the data warehouse is not needed, locking needs are minimal, only a very basic teleprocessing interface is required, and so forth. Nevertheless, there are a fair number of technological requirements for the data warehouse. This chapter outlines some of these.

Managing Large Amounts of Data

Prior to data warehousing the terms terabytes and petabytes were unknown; data capacity was measured in megabytes and gigabytes. After data warehousing the whole perception changed. Suddenly what was large one day was trifling the next. The explosion of data volume came about because the data warehouse required that both detail and history be mixed in the same environment. The issue of volumes of data is so important that it pervades all other aspects of data warehousing. With this in mind, the first and most important technological requirement for the data warehouse is the ability to manage large amounts of data, as shown in Figure 5.1. There are many approaches, and in a large warehouse environment, more than one approach will be used.

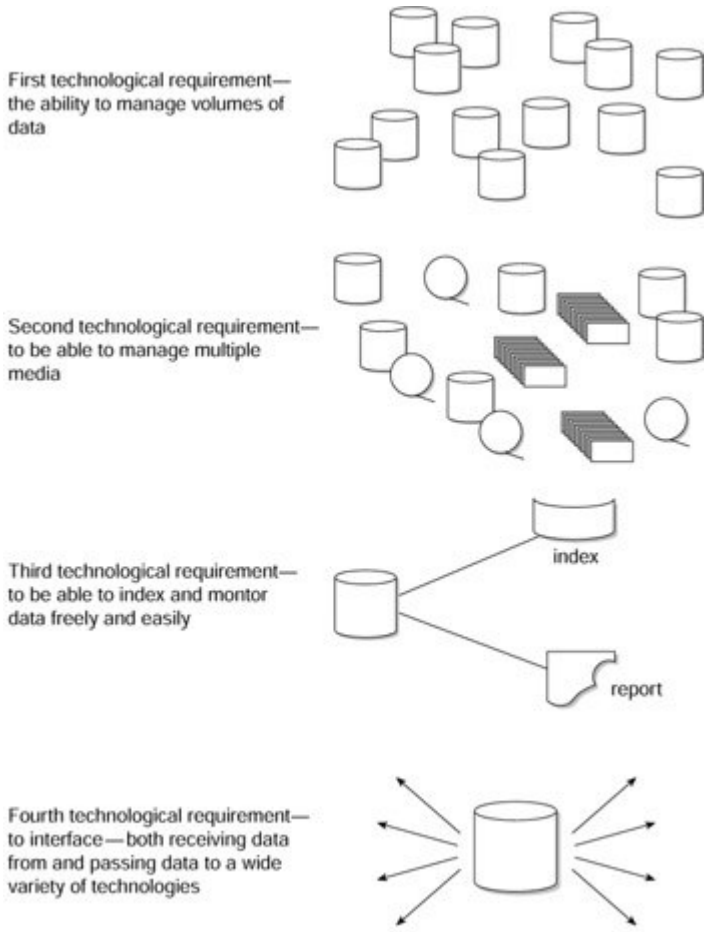

Figure 5.1: Some basic requirements for technology supporting a data warehouse.

Large amounts of data need to be managed in many ways—through flexibility of addressability of data stored inside the processor and stored inside disk storage, through indexing, through extensions of data, through the efficient management of overflow, and so forth. No matter how the data is managed, however, two fundamental requirements are evident—the ability to manage large amounts at all and the ability to manage it well. Some approaches can be used to manage large amounts of data but do so in a clumsy manner. Other approaches can manage large amounts and do so in an efficient, elegant manner. To be effective, the technology used must satisfy the requirements for both volume and efficiency.

In the ideal case, the data warehouse developer builds a data warehouse under the assumption that the technology that houses the data warehouse can handle the volumes required. When the designer has to go to extraordinary lengths in design and implementation to map the technology to the data warehouse, then there is a problem with the underlying technology. When technology is an issue, it is normal to engage more than one technology. The ability to participate in moving dormant data to overflow storage is perhaps the most strategic capability that a technology can have.

Of course, beyond the basic issue of technology and its efficiency is the cost of storage and processing.

Managing Multiple Media

In conjunction with managing large amounts of data efficiently and cost-effectively, the technology underlying the data warehouse must handle multiple storage media. It is insufficient to manage a mature data warehouse on Direct Access Storage Device (DASD) alone. Following is a hierarchy of storage of data in terms of speed of access and cost of storage:

Main memory	Very fast	Very expensive
Expanded memory	Very fast	Expensive
Cache	Very fast	Expensive
DASD	Fast	Moderate
Magnetic tape	Not fast	Not expensive
Optical disk	Not slow	Not expensive
Fiche	Slow	Cheap

The volume of data in the data warehouse and the differences in the probability of access dictates that a fully populated data warehouse reside on more than one level of storage.

Index/Monitor Data

The very essence of the data warehouse is the flexible and unpredictable access of data. This boils down to the ability to access the data quickly and easily. If data in the warehouse cannot be easily and efficiently indexed, the data warehouse will not be a success. Of course, the designer uses many practices to make data as flexible as possible, such as spreading data across different storage media and partitioning data. But the technology that houses the data must be able to support easy indexing as well. Some of the indexing techniques that often make sense are the support of secondary indexes, the support of sparse indexes, the support of dynamic, temporary indexes, and so forth. Furthermore, the cost of creating the index and using the index cannot be significant.

In the same vein, the data must be monitored at will. The cost of monitoring data cannot be so high and the complexity of monitoring data so great as to inhibit a monitoring program from being run whenever necessary. Unlike the monitoring of transaction processing, where the transactions themselves are monitored, data warehouse activity monitoring determines what data has and has not been used.

Monitoring data warehouse data determines such factors as the following:

- If a reorganization needs to be done
- If an index is poorly structured
- If too much or not enough data is in overflow
- The statistical composition of the access of the data
- Available remaining space

If the technology that houses the data warehouse does not support easy and efficient monitoring of data in the warehouse, it is not appropriate.

Interfaces to Many Technologies

Another extremely important component of the data warehouse is the ability both to receive data from and to pass data to a wide variety of technologies. Data passes into the data warehouse from the operational environment and the ODS, and from the data warehouse into data marts, DSS applications, exploration and data mining warehouses, and alternate storage. This passage must be smooth and easy. The technology supporting the data warehouse is practically worthless if there are major constraints for data passing to and from the data warehouse.

In addition to being efficient and easy to use, the interface to and from the data warehouse must be able to operate in a batch mode. Operating in an online mode is interesting but not terribly useful. Usually a period of dormancy exists from the moment that the data arrives in the operational environment until the data is ready to be passed to the data warehouse. Because of this latency, online passage of data into the data warehouse is almost nonexistent (as opposed to online movement of data into a class I ODS).

The interface to different technologies requires several considerations:

- Does the data pass from one DBMS to another easily?
- Does it pass from one operating system to another easily?
- Does it change its basic format in passage (EBCDIC, ASCII, etc.)?

Programmer/Designer Control of Data Placement

Because of efficiency of access and update, the programmer/designer must have specific control over the placement of data at the physical block/page level, as shown in Figure 5.2.

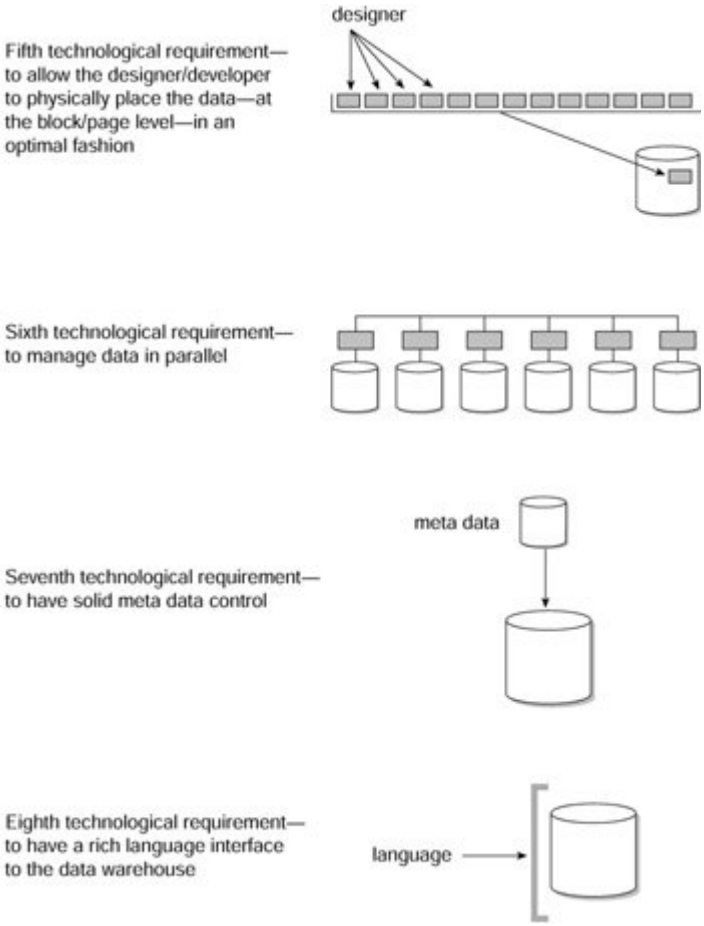

Figure 5.2: More technological requirements for the data warehouse.

The technology that houses the data in the data warehouse can place the data where it thinks is appropriate, as long as the technology can be explicitly overridden when needed. Technology that insists on the physical placement of data with no overrides from the programmer is a serious mistake.

The programmer/designer often can arrange for the physical placement of data to coincide with its usage. In doing so, many economies of resource utilization can be gained in the access of data.

Parallel Storage/Management of Data

One of the most powerful features of data warehouse data management is parallel storage/management. When data is stored and managed in a parallel fashion, the gains in performance can be dramatic. As a rule, the performance boost is inversely proportional to the number of physical devices over which the data is scattered, assuming there is an even probability of access for the data.

The entire issue of parallel storage/management of data is too complex and important to be discussed at length here, but it should be mentioned.

Meta Data Management

As mentioned in Chapter 3, for a variety of reasons, meta data becomes even more important in the data warehouse than in the classical operational environment. Meta data is vital because of the fundamental difference in the development life cycle that is associated with the data warehouse. The data warehouse operates under a heuristic, iterative development life cycle. To be effective, the user of the data warehouse must have access to meta data that is accurate and up-to-date. Without a good source of meta data to operate from, the job of the DSS analyst is much more difficult. Typically, the technical meta data that describes the data warehouse contains the following:

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

Several types of meta data need to be managed in the data warehouse: distributed meta data, central meta data, technical meta data, and business meta data. Each of these categories of meta data has its own considerations.

Language Interface

The data warehouse must have a rich language specification. The languages used by the programmer and the DSS end user to access data inside the data warehouse should be easy to use and robust. Without a robust language, entering and accessing data in the warehouse become difficult. In addition, the language used to access data needs to operate efficiently.

Typically, the language interface to the data warehouse should do the following:

- Be able to access data a set at a time
- Be able to access data a record at a time
- Specifically ensure that one or more indexes will be used in the satisfaction of a query
- Have an SQL interface
- Be able to insert, delete, or update data

There are, in fact, many different kinds of languages depending on the processing being performed. These include languages for statistical analysis of data, where data mining and exploration are done; languages for the simple access of data; languages that handle prefabricated queries; and languages that optimize on the graphic nature of the interface. Each of these languages has its own strengths and weaknesses.

Efficient Loading of Data

An important technological capability of the data warehouse is to load the data warehouse efficiently, as shown in Figure 5.3. The need for an efficient load is important everywhere, but even more so in a large warehouse.

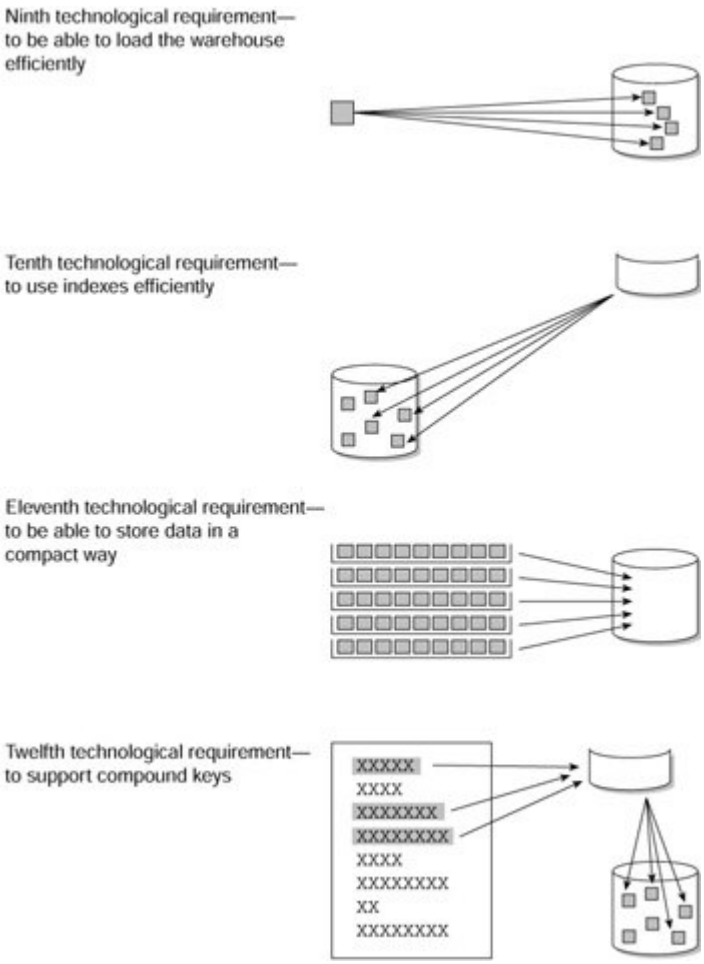

Figure 5.3: Further technological requirements.

Data is loaded into a data warehouse in two fundamental ways: a record at a time through a language interface or en masse with a utility. As a rule, loading data by means of a utility is much faster. In addition, indexes must be efficiently loaded at the same time the data is loaded. In some cases, the loading of the indexes may be deferred in order to spread the workload evenly.

As the burden of the volume of loading becomes an issue, the load is often parallelized. When this happens, the data being loaded is divided into one of several job streams. Once the input data is divided, each job stream is executed independently of the other job streams. In doing so, the elapsed time needed for loading is reduced by the number of job streams (roughly speaking).

Another related approach to the efficient loading of very large amounts of datais staging the data prior to loading. As a rule, large amounts of data are gathered into a buffer area before being processed by extract/transfer/load (ETL) software. The staged data is merged, perhaps edited, summarized, and so forth, before it passes into the ETL layer. Staging of data is needed only where the amount of data is large and the complexity of processing is high.

Efficient Index Utilization

Not only must the technology underlying the data warehouse be able to easily support the creation and loading of new indexes, but those indexes must be able to be accessed efficiently. Technology can support efficient index access in several ways:

- Using bit maps
- Having multileveled indexes
- Storing all or parts of an index in main memory
- Compacting the index entries when the order of the data being indexed allows such compaction
- Creating selective indexes and range indexes

In addition to the efficient storage and scanning of the index, the subsequent access of data at the primary storage level is important. Unfortunately, there are not nearly as many options for optimizing the access of primary data as there are for the access of index data.

Compaction of Data

The very essence of success in the data warehouse environment is the ability to manage large amounts of data. Central to this goal is the ability to compact data. Of course, when data is compacted it can be stored in a minimal amount of space. In addition, when data can be stored in a small space, the access of the data is very efficient. Compaction of data is especially relevant to the data warehouse environment because data in the warehouse environment is seldom updated once inserted in the warehouse. The stability of warehouse data minimizes the problems of space management that arise when tightly compacted data is being updated.

Another advantage is that the programmer gets the most out of a given I/O when data is stored compactly. Of course, there is always the corresponding issue of decompaction of data on access. While it is true that decompaction requires overhead, the overhead is measured in CPU resources, not I/O resources. As a rule, in the data warehouse environment, I/O resources are much more scarce than CPU resources, so decompaction of data is not a major issue.

Compound Keys

A simple but important technological requirement of the data warehouse environment is the ability to support compound keys. Compound keys occur everywhere in the data warehouse environment, primarily because of the time variancy of data warehouse data and because key/foreign key relationships are quite common in the atomic data that makes up the data warehouse.

Variable-Length Data

Another simple but vital technological requirement of the data warehouse environment is the ability to manage variable-length data efficiently, as seen in Figure 5.4. Variable-length data can cause tremendous performance problems when it is constantly being updated and changed. Where variable-length data is stable, as in the case of a data warehouse, there is no inherent performance problem.

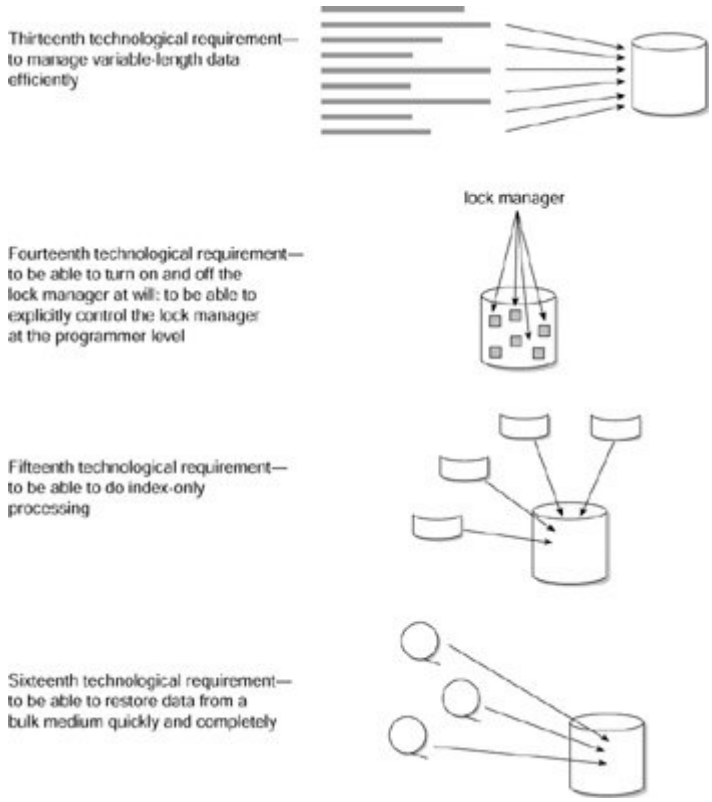

Figure 5.4: Still more technological requirements for the data warehouse.

In addition, because of the variety of data found in the data warehouse, variable-length structuring of data must be supported.

Lock Management

A standard part of database technology is the lock manager, which ensures that two or more people are not updating the same record at the same time. But update is not done in the data warehouse; instead, data is stored in a series of snapshot records. When a change occurs a new snapshot record is added, rather than an update being done.

One of the effects of the lock manager is that it consumes a fair amount of resources, even when data is not being updated. Merely turning the lock manager on requires overhead. Therefore, to streamline the data warehouse environment, being able to selectively turn the lock manager off and on is necessary.

Index-Only Processing

A fairly standard database management system feature is the ability to do index-only processing. On many occasions, it is possible to service a request by simply looking in an index (or indexes)—without going to the primary source of data. This is, of course, much more efficient. Not all DBMSs, though, are intelligent enough to know that a request can be satisfied in the index.

Technology that is optimal for the data warehouse environment looks for data in the indexes exclusively if such a request can be formulated and/or allow the query user to specify that such an index query has been specified. The DBMS technology must offer the DSS end user

the option of specifying that if an index query can be executed, the query be satisfied in that manner.

Fast Restore

A simple but important technological feature of the data warehouse environment is the ability to quickly restore a data warehouse table from non-DASD storage. When a restore can be done from secondary storage, enormous savings may be possible. Without the ability to restore data quickly from secondary storage, the standard practice is to double the amount of DASD and use one-half of the DASD as a recovery/restore repository.

The quick-restore capability must be able to restore both full databases and partial databases. The size of the data found in the data warehouse mandates that only partial databases be able to be recovered.

In addition, the DBMS needs to sense that an error has occurred in as automated a manner as possible. Leaving the detection of data corruption to the end user is a very crude way to process. Another useful technology is the ability to create diagnostic tools to determine exactly what data has been corrupted. The diagnostic tool must operate within huge amounts of data.

Other Technological Features

The features discussed here are only the most important. Many others support data warehousing, but they are too numerous to mention here.

It is noteworthy that many other features of DBMS technology found in the classical transaction processing DBMS play only a small role (if they play a role at all) in the support of the data warehouse environment. Some of those features include the following:

- Transaction integrity
- High-speed buffering
- Row/page-level locking
- Referential integrity
- VIEWs of data

Indeed, whenever a transaction-based DBMS is used in the data warehouse environment, it is desirable to turn off such features, as they interfere with the efficient processing of data inside the data warehouse.

DBMS Types and the Data Warehouse

With the advent of data warehousing and the recognition of DSS as an integral part of the modern information systems infrastructure, a new class of DBMS has arisen. This class can be called a data warehouse-specific database management system. The data warehouse-specific DBMS is optimized for data warehousing and DSS processing.

Prior to data warehousing was transaction processing, and DBMSs supported the needs of this processing type. Processing in the data warehouse, though, is quite different. Data warehouse processing can be characterized as load-and-access. Data is integrated, transformed, and loaded into the data warehouse from the operational legacy environment and the ODS. Once in the data warehouse, the integrated data is accessed and analyzed there. Update is not normally done in the data warehouse once the data is loaded. If corrections or adjustments need to be made to the data warehouse, they are made at off

hours, when no analysis is occurring against the data warehouse data. In addition, such changes are made by including a more current snapshot of data.

Another important difference between classical transaction processing database environments and the data warehouse environment is that the data warehouse environment tends to hold much more data, measured in terabytes and petabytes, than classical transaction processing databases under a general-purpose DBMSs. Data warehouses manage massive amounts of data because they contain the following:

- Granular, atomic detail
- Historical information
- Summary as well as detailed data

In terms of basic data management capability, data warehouses are optimized around a very different set of parameters than standard operational DBMSs.

The first and most important difference between a classical, general-purpose DBMS and a data warehouse-specific DBMS is how updates are performed. A classical, general-purpose DBMS must be able to accommodate record-level, transaction-based updates as a normal part of operations. Because record-level, transaction-based updates are a regular feature of the general-purpose DBMS, the general-purpose DBMS must offer facilities for such items as the following:

- Locking
- COMMITs
- Checkpoints
- Log tape processing
- Deadlock
- Backout

Not only do these features become a normal part of the DBMS, they consume a tremendous amount of overhead. Interestingly, the overhead is consumed even when it isn't being used. In other words, at least some update and locking overhead—depending on the DBMS—is required by a general-purpose DBMS even when read-only processing is being executed. Depending on the general-purpose DBMS, the overhead required by update can be minimized, but it cannot be completely eliminated. For a data warehouse-specific DBMS, there is no need for any of the overhead of update.

A second major difference between a general-purpose DBMS and a data warehouse-specific DBMS regards basic data management. For a general-purpose DBMS, data management at the block level includes space that is reserved for future block expansion at the moment of update or insertion. Typically, this space is referred to as *freespace*. For a general-purpose DBMS, freespace may be as high as 50 percent. For a data warehouse-specific DBMS, freespace always equals 0 percent because there is no need for expansion in the physical block, once loaded; after all, update is not done in the data warehouse environment. Indeed, given the amount of data to be managed in a data warehouse, it makes no sense to reserve vast amounts of space that may never be used.

Another relevant difference between the data warehouse and the general-purpose environment that is reflected in the different types of DBMS is indexing. A general-purpose DBMS environment is restricted to a finite number of indexes. This restriction exists because as updates and insertions occur, the indexes themselves require their own space and their own data management. In a data warehouse environment where there is no update and there is a need to optimize access of data, there is a need (and an opportunity) for many indexes. Indeed, a much more robust and sophisticated indexing structure can be employed for data warehousing than for operational, update-oriented databases.

Beyond indexing, update, and basic data management at the physical block level are some other very basic differences between the data management capabilities and philosophies of general-purpose transaction processing DBMSs and data warehouse-specific DBMSs. Perhaps the most basic difference is the ability to physically organize data in an optimal fashion for different kinds of access. A general-purpose DBMS typically physically organizes data for optimal transaction access and manipulation. Organizing in this fashion allows many different types of data to be gathered according to a common key and efficiently accessed in one or two I/Os. Data that is optimal for informational access usually has a very different physical profile. Data that is optimal for informational access is organized so that many different occurrences of the same type of data can be accessed efficiently in one or two physical I/Os.

Data can be physically optimized for transaction access or DSS access, but not both at the same time. A general-purpose, transaction-based DBMS allows data to be optimized for transaction access, and a data warehouse-specific DBMS allows data to be physically optimized for DSS access and analysis.

Changing DBMS Technology

An interesting consideration of the information warehouse is changing the DBMS technology after the warehouse has already been populated. Such a change may be in order for several reasons:

- DBMS technologies may be available today that simply were not an option when the data warehouse was first populated.
- The size of the warehouse has grown to the point that a new technological approach is mandated.
- Use of the warehouse has escalated and changed to the point that the current warehouse DBMS technology is not adequate.
- The basic DBMS decision must be revisited from time to time.

Should the decision be made to go to a new DBMS technology, what are the considerations? A few of the more important ones follow:

- Will the new DBMS technology meet the foreseeable requirements?
- How will the conversion from the older DBMS technology to the newer DBMS technology be done?
- How will the transformation programs be converted?

Of all of these considerations, the last is the most vexing. Trying to change the transformation programs is a complex task in the best of circumstances.

The fact remains that once a DBMS has been implemented for a data warehouse, change at a later point in time is a possibility. Such was never the case in the world of transaction processing; once a DBMS had been implemented, that DBMS stayed as long as the transactions were being run.

Multidimensional DBMS and the Data Warehouse

One of the technologies often discussed in the context of the data warehouse is multidimensional database management systems processing (sometimes called OLAP processing). Multidimensional database management systems, or data marts, provide an information system with the structure that allows an organization to have very flexible access to data, to slice and dice data any number of ways, and to dynamically explore the relationship between summary and detail data. Multidimensional DBMSs offer both flexibility and control to the end user, and as such they fit well in a DSS environment. A very

interesting and complementary relationship exists between multidimensional DBMSs and the data warehouse, as shown in Figure 5.5.

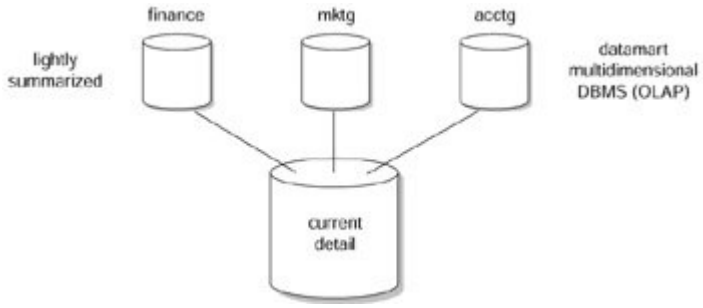

Figure 5.5: The classical structure of the data warehouse and how current detail data and departmental data (or multidimensional DBMS, data mart) data fit together.

The detailed data housed in a data warehouse provides a very robust and convenient source of data for the multidimensional DBMS. Data flows from the data warehouse into the multidimensional DBMS on a regular basis as the multidimensional DBMS needs to be periodically refreshed. Because legacy application data is integrated as it enters the data warehouse, the multidimensional DBMS does not need to extract and integrate the data it operates on from the operational environment. In addition, the data warehouse houses data at its lowest level, providing "bedrock" data for the lowest level of analysis that anyone using the multidimensional DBMS would ever want.

Though tempting to think that multidimensional DBMS technology should be the database technology for the data warehouse, in all but the most unusual cases, this is a mistake. The properties that make multidimensional DBMS technology optimal for what it does are not the properties of primary importance for the data warehouse, and the properties that are the most important in the data warehouse are not those found in multidimensional DBMS technology.

Consider the differences between the multidimensional DBMS and the data warehouse:

- The data warehouse holds massive amounts of data; the multidimensional DBMS holds at least an order of magnitude less data.
- The data warehouse is geared for a limited amount of flexible access; the multidimensional DBMS is geared for very heavy and unpredictable access and analysis of data.
- The data warehouse contains data with a very lengthy time horizon—from 5 to 10 years; the multidimensional DBMS holds a much shorter time horizon of data.
- The data warehouse allows analysts to access its data in a constrained fashion; the multidimensional DBMS allows unfettered access.
- Instead of the data warehouse being housed in a multidimensional DBMS, the multidimensional DBMS and the data warehouse enjoy a complementary relationship.

One of the interesting features of the relationship between the data warehouse can contain a very fine degree of detail, which is lightly summarized as it is passed up to the multidimensional DBMS. Once in the multidimensional DBMS, the data can be further summarized. In such a fashion the multidimensional DBMS can house all but the most detailed level of data. The analyst using the multidimensional DBMS can drill down in a flexible and efficient manner over all the different levels of data found in it. Then, if needed, the analyst can actually drill down to the data warehouse. By marrying the data warehouse and the multidimensional DBMS in such a manner, the DSS analyst gets the best of both worlds. The DSS analyst enjoys the efficiency of operating most of the time in the world of

the multidimensional DBMS while at the same time being able to drill down to the lowest level of detail.

Another advantage is that summary information may be calculated and collected in the multidimensional DBMS and then stored in the data warehouse. When this is done, the summary data can be stored in the data warehouse for a much longer time than if it were stored in the multidimensional DBMS.

There is still another way that the multidimensional DBMS and data warehouse worlds are complementary. The multidimensional DBMS houses data over a modest length of time—say 12 to 15 months, depending on the application. The data warehouse houses data over a much longer time—5 to 10 years. In such a manner, the data warehouse becomes a source of research for multidimensional DBMS analysts. Multidimensional DBMS analysts have the luxury of knowing that huge amounts of data are available if needed, but they do not have to pay the price of storing all that data in their environment.

Multidimensional DBMSs come in several flavors. Some multidimensional DBMSs operate on a foundation of relational technology, and some operate on a technological foundation optimal for "slicing and dicing" the data, where data can be thought of as existing in multidimensional cubes. The latter technological foundation is sometimes called a cube or OLAP foundation.

Both foundations can support multidimensional DBMS data marts. But there are some differences between the two types of technological foundations:

The relational foundation for multidimensional DBMS data marts:

- Strengths:
  - Can support a lot of data
  - Can support dynamic joining of data
  - Has proven technology
  - Is capable of supporting general-purpose update processing
  - If there is no known pattern of usage of data, then the relational structure is as good as any other
- Weaknesses:
  - Has performance that is less than optimal
  - Cannot be purely optimized for access processing

The cube foundation for multidimensional DBMS data marts:

- Strengths:
  - Performance that is optimal for DSS processing
  - Can be optimized for very fast access of data
  - If pattern of access of data is known, then the structure of data can be optimized
  - Can easily be sliced and diced
  - Can be examined in many ways
- Weaknesses:
  - Cannot handle nearly as much data as a standard relational format
  - Does not support general-purpose update processing
  - May take a long time to load
  - If access is desired on a path not supported by the design of the data, the structure is not flexible
  - Questionable support for dynamic joins of data

Multidimensional DBMS (OLAP) is a technology, while the data warehouse is an architectural infrastructure, and a symbiotic relationship exists between the two. In the normal case, the data warehouse serves as a foundation for the data that will flow into the

multidimensional DBMS—feeding selected subsets of the detailed data into the multidimensional DBMS where it is summarized and otherwise aggregated. But in some circles there is the notion that multidimensional DBMSs do not need a data warehouse for their foundation of data.

Without a data warehouse serving as the foundation for the multidimensional DBMS, the data flowing into the multidimensional DBMS comes directly from the older, legacy applications environment. Figure 5.6 shows the flow of data from the legacy environment directly to the multidimensional DBMS. The design is appealing because it is straightforward and easily achieved. A programmer can immediately start to work on building it.

Unfortunately, some major pitfalls in the architecture, as suggested by Figure 5.6, are not immediately apparent. For a variety of reasons, it makes sense to feed the multidimensional DBMS environment from the current level of detail of the data warehouse, rather than feeding it directly from the legacy applications operational environment.

Figure 5.6: Building the multidimensional DBMS data mart from an application with no current detail.

Figure 5.7 illustrates the feeding of the multidimensional DBMS environment from the current level of detail of the data warehouse environment. Old, legacy operational data is integrated and transformed as it flows into the data warehouse.

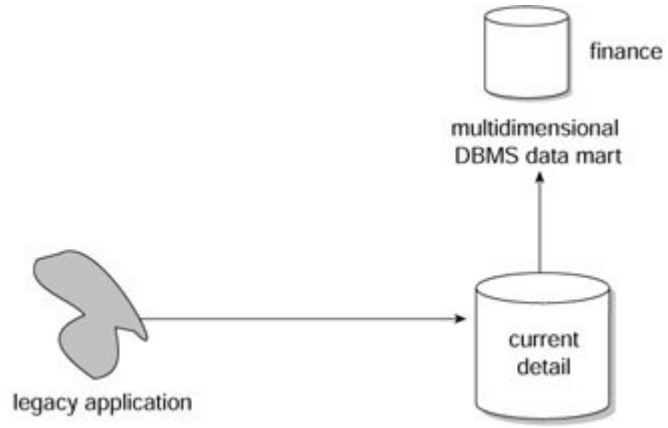

Figure 5.7: The flow of data from the application environment to the current level detail to the multidimensional DBMS data mart.

Once in the data warehouse, the integrated data is stored in the current level of detailed data. From this level, the multidimensional DBMS is fed.

At first glance, there may not appear to be substantive differences between the architectures shown in Figure 5.6 and Figure 5.7. In fact, putting data first into a data warehouse may even appear to be a wasted effort. However, there is a very good reason why integrating data into the data warehouse is the first step in creating the multidimensional DBMS.

Consider that under normal conditions a corporation will want to build multiple multidimensional DBMSs. Finance will want its multidimensional DBMS, as will accounting. Marketing, sales, and other departments will want their own multidimensional DBMSs. Because multiple multidimensional DBMSs will be in the corporation, the scenario shown in Figure 5.6 becomes much more complex. In Figure 5.8, Figure 5.6 has been expanded into a realistic scenario where there are multiple multidimensional DBMSs being directly and individually fed from the legacy systems environment.

Figure 5.8 shows that multiple multidimensional DBMSs are being fed directly from the same legacy applications. So, what is so wrong with this architecture?

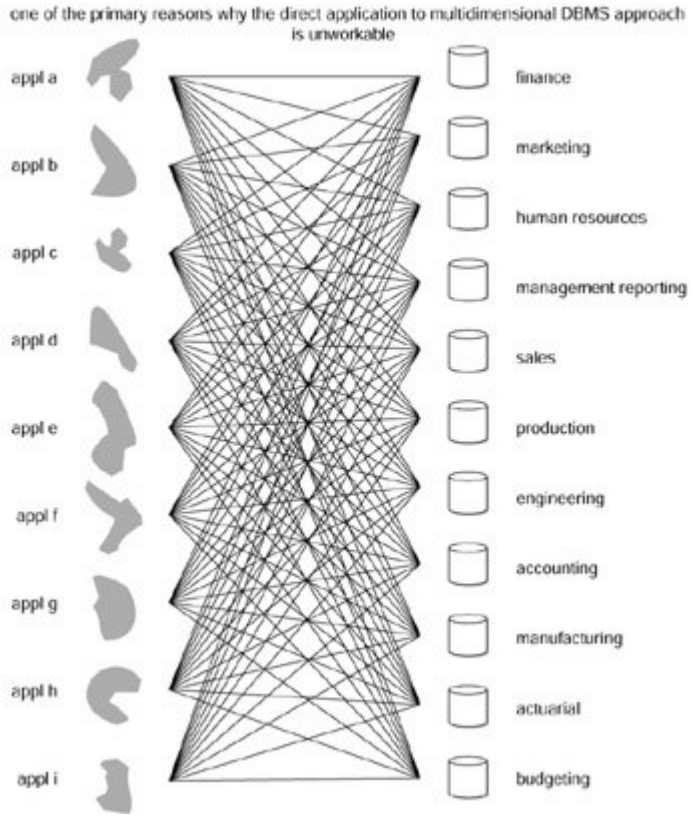

Figure 5.8: There are many applications, and there are many data marts. An interface application is needed between each occurrence. The result of bypassing the current level of detail is an unmanageable "spider web."

The problems are as follows:

- The amount of development required in extraction is enormous. Each different departmental multidimensional DBMS must have its own set of extraction programs developed for it on a customized basis. There is a tremendous overlap of extract processing. The amount of wasted development work is enormous. When the multidimensional DBMSs are fed from the data warehouse, only one set of integration and transformation programs is needed.
- There is no integrated foundation when the multidimensional DBMSs are fed directly from the legacy systems environment. Each departmental multidimensional DBMS has its own interpretation as to how different applications should be integrated. Unfortunately, the way one department integrates data is most likely not the way another department integrates the same data. The result is that there is no single

integrated, definitive source of data. Conversely, when the data warehouse is built, there is a single, definitive, integrated source of data that can be built upon.

- The amount of development work required for maintenance is enormous. A single change in an old legacy application ripples through many extraction programs. The change must be accommodated wherever there is an extraction program, and there are many. With a data warehouse, the effect of change is minimized because a minimal number of programs must be written to manage the interface between the legacy environment and the data warehouse.
- The amount of hardware resources consumed is great. The same legacy data is sequentially and repeatedly passed for each extraction process for each department. In the case of the data warehouse, the legacy data is passed only once to refresh the data in the data warehouse.
- The complexity of moving data directly from the legacy environment to the multidimensional DBMS environment precludes effective meta data management and control. With the data warehouse, capturing and managing meta data are both straightforward.
- The lack of reconcilability of data is an issue. When a difference in opinion exists among various departments, each having its own multidimensional DBMS, there is no easy resolution. With a data warehouse, resolution of conflicts is natural and easy.
- Each time a new multidimensional DBMS environment must be built, it must be built from the legacy environment, and the amount of work required is considerable. When a foundation of data is in a data warehouse, however, building a new multidimensional DBMS environment is quick and easy.

When an organization takes a short-term approach, justifying the data warehouse is hard to do. The long-term costs of building many multidimensional database environments is very high. When an organization takes a long-term view and builds a data warehouse, the long-term total cost of data warehousing and data marts drops significantly.

Data Warehousing across Multiple Storage Media

One interesting aspect of a data warehouse is the dual environments often created when a large amount of data is spread across more than one storage media. One processing environment is the DASD environment where online/interactive processing is done. The other processing environment is often a tape or mass store environment, which has essentially different features. Logically, the two environments combine to form a single data warehouse. Physically, however, the two environments are very different. In many cases, the underlying technology that supports the DASD environment is not the same technology that supports the mass store environment. Mixing technologies in the data warehouse environment is normal and natural when done this way.

However, there is another way that technology can be split that is not normal or natural. It is conceivable that the data warehouse environment—the DASD portion—is split over more than one technology. In other words, part of the DASD-based data warehouse resides on one vendor's technology and another part of the data warehouse resides on another vendor's database technology. If the split is deliberate and part of a larger distributed data warehouse, such a split is just fine. But if the split occurs for political or historical reasons, splitting part of a data warehouse onto different vendor platforms is not advisable.

Meta Data in the Data Warehouse Environment

The role of meta data in the data warehouse environment is very different from the role of meta data in the operational environment. In the operational environment, meta data is treated almost as an afterthought and is relegated to the same level of importance as documentation. Meta data in the data warehouse environment takes on a very enhanced

role. The importance of its role in the data warehouse environment is illustrated in Figure 5.9. Two different communities are served by operational meta data and data warehouse meta data. Operational meta data is used by the IT professional. For years, the IT professional has used meta data casually. The IT professional is computer-literate and is able to find his or her way around systems. The data warehouse, though, serves the DSS analyst community, and the DSS analyst is usually a professional, first and foremost. There usually is not a high degree of computer literacy in the DSS analyst community. The DSS analyst needs as much help as possible to use the data warehouse environment effectively, and meta data serves this end quite well. In addition, meta data is the first thing the DSS analyst looks at in planning how to perform informational/analytical processing. Because of the difference in the communities served and because of the role that meta data plays in the day-to-day job function, meta data is much more important in the data warehouse environment than it ever was in the operational environment.

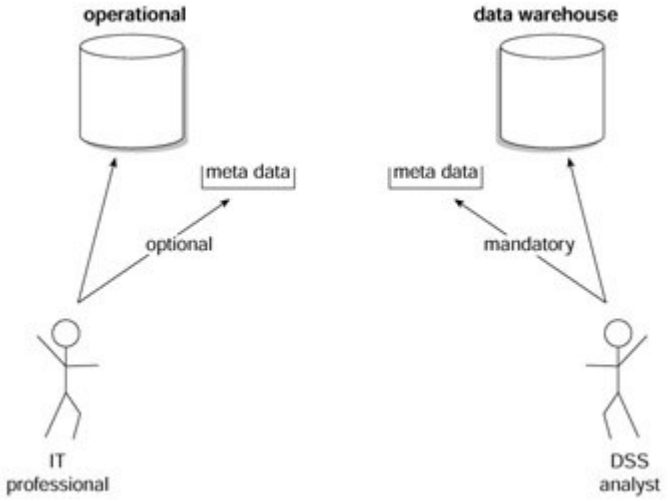

Figure 5.9: The IT professional uses meta data on a casual basis; the DSS analyst uses meta data regularly and as the first step of an analysis.

But there are other reasons why data warehouse meta data is important. One such reason concerns managing the mapping between the operational environment and the data warehouse environment. Figure 5.10 illustrates this point.

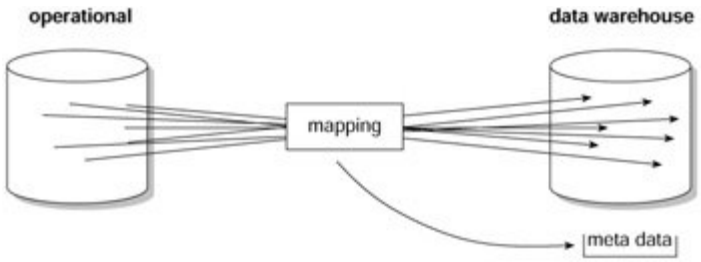

Figure 5.10: The mapping between the operational environment and the data warehouse environment is another major reason for the need for meta data; without the mapping, controlling the interface is extremely difficult.

Data undergoes a significant transformation as it passes from the operational environment to the data warehouse environment. Conversion, filtering, summarization, and structural changes all occur. There is a need to keep careful track of the transformation, and the meta data in the data warehouse is the ideal place to do so. The importance of keeping a careful

record of the transformation is highlighted by the events that occur when a manager needs to trace data from the data warehouse back to its operational source (the ultimate in the drill-down process!). In this case, the record of the transformation describes exactly how to get from the data warehouse to the operational source of data.

Yet another important reason for the careful management of meta data in the data warehouse environment is shown in Figure 5.11. As mentioned, data in a data warehouse exists for a lengthy timespan—from 5 to 10 years. Over a 5-to-10-year time span it is absolutely normal for a data warehouse to change its structure. Keeping track of the changing structure of data over time is a natural task for the meta data in the data warehouse.

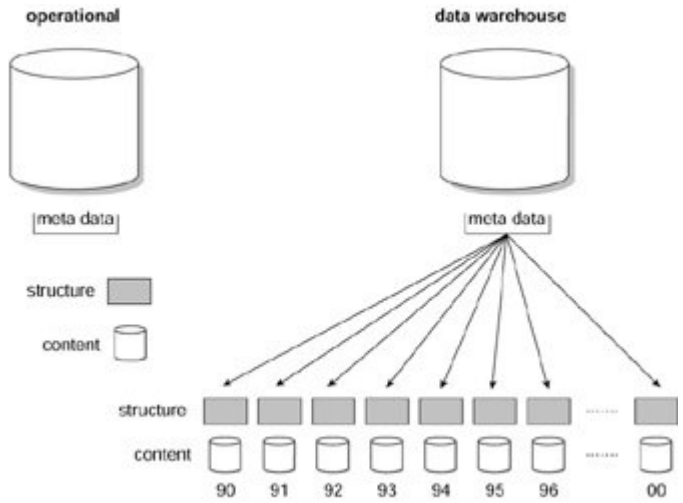

Figure 5.11: The data warehouse contains data over a long period of time and must manage multiple structures/definitions of data. The operational environment assumes that there is only a single correct definition of data at any one time.

Contrast the notion that there will be many structures of data over time in the data warehouse environment with the meta data found in the operational environment. In the operational environment it is assumed that at any one moment, there is one and only one correct definition of the structure of data.

Context and Content

In the past, classical operational information systems have focused their attention on the very current data of a corporation. In the operational world, the emphasis is on how much an account balance is, right now or how much is in inventory, right now or what the status of a shipment is, right now. Of course, every organization needs to know about current information. But there is real value in looking at information over time, as is possible with data warehousing. For example, trends become apparent that simply are not observable when looking at current information. One of the most important defining characteristics of the data warehouse is this ability to store, manage, and access data over time.

With the lengthy spectrum of time that is part of a data warehouse comes a new dimension of data-context. To explain the importance of contextual information, an example is in order.

Suppose a manager asks for a report from the data warehouse for 1995. The report is generated, and the manager is pleased. In fact, the manager is so pleased that a similar report for 1990 is requested. Because the data warehouse carries historical information,

such a request is not hard to accommodate. The report for 1990 is generated. Now the manager holds the two reports—one for 1995 and one for 1990—in his hands and declares that the reports are a disaster.

The data warehouse architect examines the reports and sees that the financial statement for 1995 shows $50 million in revenue, while the report for 1990 shows a value of $10,000 for the same category. The manager declares that there is no way that any account or category could have increased in value that much in five years' time.

Before giving up, the data warehouse architect points out to the manager that there are other relevant factors that do not show up in the report. In 1990, there was a different source of data than in 1995. In 1990, the definition of a product was not the same as in 1995. In 1990, there were different marketing territories than in 1995. In 1990, there were different calculations, such as for depreciation, than in 1995. In addition were many different external considerations, such as a difference in inflation, taxation, economic forecasts, and so forth. Once the context of the reports is explained to the manager, the contents now appear to be quite acceptable.

In this simple but common example where the contents of data stand naked over time, the contents by themselves are quite inexplicable and unbelievable. When context is added to the contents of data over time, the contents and the context become quite enlightening.

To interpret and understand information over time, a whole new dimension of context is required. While content of information remains important, the comparison and understanding of information over time mandates that context be an equal partner to content. And in years past, context has been an undiscovered, unexplored dimension of information.

Three Types of Contextual Information

Three levels of contextual information must be managed:

- Simple contextual information
- Complex contextual information
- External contextual information

Simple contextual information relates to the basic structure of data itself, and includes such things as these:

- The structure of data
- The encoding of data
- The naming conventions used for data
- The metrics describing the data, such as:
  - How much data there is
  - How fast the data is growing
  - What sectors of the data are growing
  - How the data is being used

Simple contextual information has been managed in the past by dictionaries, directories, system monitors, and so forth. Complex contextual information describes the same data as simple contextual information, but from a different perspective. This type of information addresses such aspects of data as these:

- Product definitions
- Marketing territories
- Pricing
- Packaging
- Organization structure
- Distribution

Complex contextual information is some of the most useful and, at the same time, some of the most elusive information there is to capture. It is elusive because it is taken for granted and is in the background. It is so basic that no one thinks to define what it is or how it changes over time. And yet, in the long run, complex contextual information plays an extremely important role in understanding and interpreting information over time.

External contextual information is information outside the corporation that nevertheless plays an important role in understanding information over time. Some examples of external contextual information include the following:

- Economic forecasts:
  - Inflation
  - Financial trends
  - Taxation
  - Economic growth
- Political information
- Competitive information
- Technological advancements
- Consumer demographic movements

External contextual information says nothing directly about a company but says everything about the universe in which the company must work and compete. External contextual information is interesting both in terms of its immediate manifestation and its changes over time. As with complex contextual information, there is very little organized attempt to capture and measure this information. It is so large and so obvious that it is taken for granted, and it is quickly forgotten and difficult to reconstruct when needed.

Capturing and Managing Contextual Information

Complex and external contextual types of information are hard to capture and quantify because they are so unstructured. Compared to simple contextual information, external and complex contextual types of information are very amorphous. Another mitigating factor is that contextual information changes quickly. What is relevant one minute is passé the next. It is this constant flux and the amorphous state of external and complex contextual information that makes these types of information so hard to systematize.

Looking at the Past

One can argue that the information systems profession has had contextual information in the past. Dictionaries, repositories, directories, and libraries are all attempts at the management of simple contextual information. For all the good intentions, there have been some notable limitations in these attempts that have greatly short-circuited their effectiveness. Some of these shortcomings are as follows:

- The information management attempts were aimed at the information systems developer, not the end user. As such, there was very little visibility to the end user. Consequently, the end user had little enthusiasm or support for something that was not apparent.
- Attempts at contextual management were passive. A developer could opt to use or not use the contextual information management facilities. Many chose to work around those facilities.
- Attempts at contextual information management were in many cases removed from the development effort. In case after case, application development was done in 1965, and the data dictionary was done in 1985. By 1985, there were no more development dollars. Furthermore, the people who could have helped the most in organizing and defining simple contextual information were long gone to other jobs or companies.

- Attempts to manage contextual information were limited to only simple contextual information. No attempt was made to capture or manage external or complex contextual information.

Refreshing the Data Warehouse

Once the data warehouse is built, attention shifts from the building of the data warehouse to its day-to-day operations. Inevitably, the discovery is made that the cost of operating and maintaining a data warehouse is high, and the volume of data in the warehouse is growing faster than anyone had predicted. The widespread and unpredictable usage of the data warehouse by the end-user DSS analyst causes contention on the server managing the warehouse. Yet the largest unexpected expense associated with the operation of the data warehouse is the periodic refreshment of legacy data. What starts out as an almost incidental expense quickly turns very significant.

The first step most organizations take in the refreshment of data warehouse data is to read the old legacy databases. For some kinds of processing and under certain circumstances, directly reading the older legacy files is the only way refreshment can be achieved, for instance, when data must be read from different legacy sources to form a single unit that is to go into the data warehouse. In addition, when a transaction has caused the simultaneous update of multiple legacy files, a direct read of the legacy data may be the only way to refresh the warehouse.

As a general-purpose strategy, however, repeated and direct reads of the legacy data are a very costly. The expense of direct legacy database reads mounts in two ways. First, the legacy DBMS must be online and active during the read process. The window of opportunity for lengthy sequential processing for the legacy environment is always limited. Stretching the window to refresh the data warehouse is never welcome. Second, the same legacy data is needlessly passed many times. The refreshment scan must process 100 percent of a legacy file when only 1 or 2 percent of the legacy file is actually needed. This gross waste of resources occurs each time the refreshment process is done. Because of these inefficiencies, repeatedly and directly reading the legacy data for refreshment is a strategy that has limited usefulness and applicability.

A much more appealing approach is to trap the data in the legacy environment as it is being updated. By trapping the data, full table scans of the legacy environment are unnecessary when the data warehouse must be refreshed. In addition, because the data can be trapped as it is being updated, there is no need to have the legacy DBMS online for a long sequential scan. Instead, the trapped data can be processed offline.

Two basic techniques are used to trapp data as update is occurring in the legacy operational environment. One technique is called *data replication;* the other is called *change data capture*, where the changes that have occurred are pulled out of log or journal tapes created during online update. Each approach has its pros and cons.

Replication requires that the data to be trapped be identified prior to the update. Then, as update occurs, the data is trapped. A trigger is set that causes the update activity to be captured. One of the advantages of replication is that the process of trapping can be selectively controlled. Only the data that needs to be captured is, in fact, captured. Another advantage of replication is that the format of the data is “clean” and well defined. The content and structure of the data that has been trapped are well documented and readily understandable to the programmer. The disadvantages of replication are that extra I/O is incurred as a result of trapping the data and because of the unstable, ever-changing nature of the data warehouse, the system requires constant attention to the definition of the parameters and triggers that control trapping. The amount of I/O required is usually

nontrivial. Furthermore, the I/O that is consumed is taken out of the middle of the high-performance day, at the time when the system can least afford it.

The second approach to efficient refreshment is changed data capture (CDC). One approach to CDC is to use the log tape to capture and identify the changes that have occurred throughout the online day. In this approach, the log or journal tape is read. Reading a log tape is no small matter, however. Many obstacles are in the way, including the following:

- The log tape contains much extraneous data.
- The log tape format is often arcane.
- The log tape contains spanned records.
- The log tape often contains addresses instead of data values.
- The log tape reflects the idiosyncracies of the DBMS and varies widely from one DBMS to another.

The main obstacle in CDC, then, is that of reading and making sense out of the log tape. But once that obstacle is passed, there are some very attractive benefits to using the log for data warehouse refreshment. The first advantage is efficiency. Unlike replication processing, log tape processing requires no extra I/O. The log tape will be written regardless of whether it will be used fordata warehouse refreshment. Therefore, no incremental I/O is necessary. The second advantage is that the log tape captures all update processing. There is no need to go back and redefine parameters when a change is made to the data warehouse or the legacy systems environment. The log tape is as basic and stable as you can get.

There is a second approach to CDC: lift the changed data out of the DBMS buffers as change occurs. In this approach the change is reflected immediately. So reading a log tape becomes unnecessary, and there is a time-savings from the moment a change occurs to when it is reflected in the warehouse. However, because more online resources are required, including system software sensitive to changes, there is a performance impact. Still, this direct buffer approach can handle large amounts of processing at a very high speed.

The progression described here mimics the mindset of organizations as they mature in their understanding and operation of the data warehouse. First, the organization reads legacy databases directly to refresh its data warehouse. Then it tries replication. Finally, the economics and the efficiencies of operation lead it to CDC as the primary means to refresh the data warehouse. Along the way it is discovered that a few files require a direct read. Other files work best with replication. But for industrial-strength, full-bore, general-purpose data warehouse refreshment, CDC looms as the long-term final approach to data warehouse refreshment.

Testing

In the classical operational environment, two parallel environments are set up—one for production and one for testing. The production environment is where live processing occurs. The testing environment is where programmers test out new programs and changes to existing programs. The idea is that it is safer when programmers have a chance to see if the code they have created will work before it is allowed into the live online environment.

It is very unusual to find a similar test environment in the world of the data warehouse, for the following reasons:

- Data warehouses are so large that a corporation has a hard time justifying one of them, much less two of them.
- The nature of the development life cycle for the data warehouse is iterative. For the most part, programs are run in a heuristic manner, not in a repetitive manner. If a programmer gets something wrong in the data warehouse environment (and

programmers do all the time), the environment is set up so that the programmer simply redoes it.

The data warehouse environment then is fundamentally different from the classical production environment because, under most circumstances, a test environment is simply not needed.

Summary

Some technological features are required for satisfactory data warehouse processing. These include a robust language interface, the support of compound keys and variable-length data, and the abilities to do the following:

- Manage large amounts of data.
- Manage data on a diverse media.
- Easily index and monitor data.
- Interface with a wide number of technologies.
- Allow the programmer to place the data directly on the physical device.
- Store and access data in parallel.
- Have meta data control of the warehouse.
- Efficiently load the warehouse.
- Efficiently use indexes.
- Store data in a compact way.
- Support compound keys.
- Selectively turn off the lock manager.
- Do index-only processing.
- Quickly restore from bulk storage.

Additionally, the data architect must recognize the differences between a transaction-based DBMS and a data warehouse-based DBMS. A transaction-based DBMS focuses on the efficient execution of transactions and update. A data warehouse-based DBMS focuses on efficient query processing and the handling of a load and access workload.

Multidimensional OLAP technology is suited for data mart processing and not data warehouse processing. When the data mart approach is used as a basis for data warehousing, many problems become evident:

- The number of extract programs grows large.
- Each new multidimensional database must return to the legacy operational environment for its own data.
- There is no basis for reconciliation of differences in analysis.
- A tremendous amount of redundant data among different multidimensional DBMS environments exists.

Finally, meta data in the data warehouse environment plays a very different role than meta data in the operational legacy environment.

Chapter 6: The Distributed Data Warehouse

Overview

Most organizations build and maintain a single centralized data warehouse environment. This setup makes sense for many reasons:

- The data in the warehouse is integrated across the corporation, and an integrated view is used only at headquarters.
- The corporation operates on a centralized business model.
- The volume of data in the data warehouse is such that a single centralized repository of data makes sense.
- Even if data could be integrated, if it were dispersed across multiple local sites, it would be cumbersome to access.

In short, the politics, the economics, and the technology greatly favor a single centralized data warehouse. Still, in a few cases, a distributed data warehouse makes sense, as we'll see in this chapter.

Types of Distributed Data Warehouses

The three types of distributed data warehouses are as follows:

- Business is distributed geographically or over multiple, differing product lines. In this case, there is what can be called a local data warehouse and a global data warehouse. The local data warehouse represents data and processing at a remote site, and the global data warehouse represents that part of the business that is integrated across the business.
- The data warehouse environment will hold a lot of data, and the volume of data will be distributed over multiple processors. Logically there is a single data warehouse, but physically there are many data warehouses that are all tightly related but reside on separate processors. This configuration can be called the technologically distributed data warehouse.
- The data warehouse environment grows up in an uncoordinated manner—first one data warehouse appears, then another. The lack of coordination of the growth of the different data warehouses is usually a result of political and organizational differences. This case can be called the independently evolving distributed data warehouse.

Each of these types of distributed data warehouse has its own concerns and considerations, which we will examine in the following sections.

Local and Global Data Warehouses

When a corporation is spread around the world, information is needed both locally and globally. The global needs for corporate information are met by a central data warehouse where information is gathered. But there is also a need for a separate data warehouse at each local organization—that is, in each country. In this case, a distributed data warehouse is needed. Data will exist both centrally and in a distributed manner.

A second case for a local/global distributed data warehouse occurs when a large corporation has many lines of business. Although there may be little or no business integration among the different vertical lines of business, at the corporate level—at least as far as finance is concerned—there is. The different lines of business may not meet anywhere else but at the balance sheet, or there may be considerable business integration, including such things as

customers, products, vendors, and the like. In this scenario, a corporate centralized data warehouse is supported by many different data warehouses for each line of business.

In some cases part of the data warehouse exists centrally (i.e., globally), and other parts of the data warehouse exist in a distributed manner (i.e., locally).

To understand when a geographically or distributed business distributed data warehouse makes sense, consider some basic topologies of processing. Figure 6.1 shows a very common processing topology.

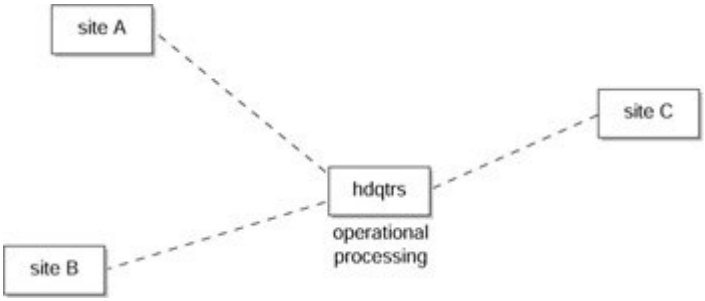

Figure 6.1: A topology of processing representative of many enterprises.

In Figure 6.1, all processing is done at the organization's headquarters. If any processing is done at the local geographically dispersed level, it is very basic, involving, perhaps, a series of dumb terminals. In this type of topology it is very unlikely that a distributed data warehouse will be necessary.

One step up the ladder in terms of sophistication of local processing is the case where basic data and transaction capture activity occurs at the local level, as shown in Figure 6.2. In this scenario, some small amount of very basic processing occurs at the local level. Once the transactions that have occurred locally are captured, they are shipped to a central location for further processing.

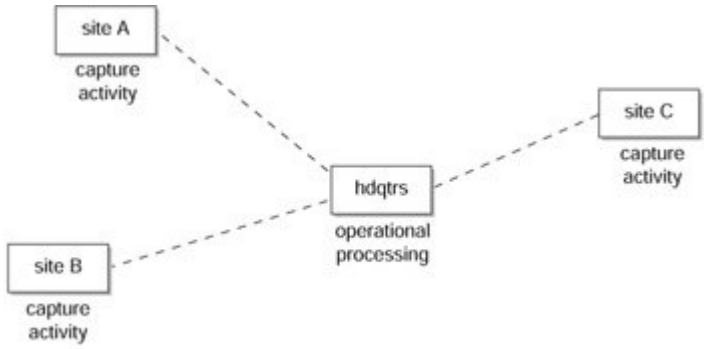

Figure 6.2: In some cases, very basic activity is done at the site level.

Under this simple topology it is very unlikely that a distributed data warehouse is needed. From a business standpoint, no great amount of business occurs locally, and decisions made locally do not warrant a data warehouse.

Now, contrast the processing topology shown in Figure 6.3 with the previous two. In Figure 6.3, a fair amount of processing occurs at the local level. Sales are made. Money is collected. Bills are paid locally. As far as operational processing is concerned, the local sites are autonomous. Only on occasion and for certain types of processing will data and activities

be sent to the central organization. A central corporate balance sheet is kept. It is for this type of organization that some form of distributed data warehouse makes sense.

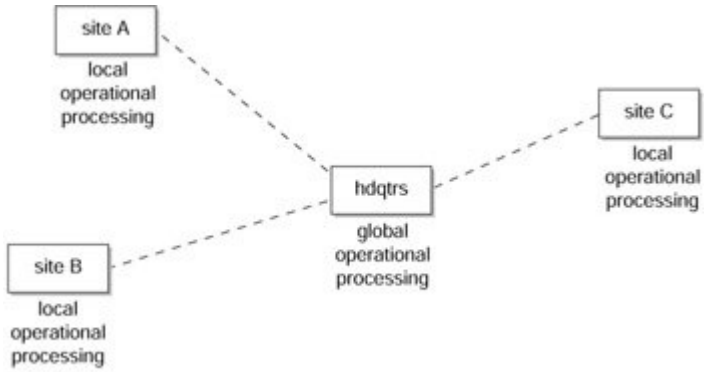

Figure 6.3: At the other end of the spectrum of the distributed data warehouse, much of the operational processing is done locally.

And then, of course, there is the even larger case where much processing occurs at the local level. Products are made. Sales forces are hired. Marketing is done. An entire mini-corporation is set up locally. Of course, the local corporations report to the same balance sheet as all other branches of the corporation. But, at the end of the day, the local organizations are effectively their own company, and there is little business integration of data across the corporation. In this case, a full-scale data warehouse at the local level is needed.

Just as there are many different kinds of distributed business models, there is more than one type of local/global distributed data warehouse, as will be discussed. It is a mistake to think that the model for the local/global distributed data warehouse is a binary proposition. Instead, there are degrees of distributed data warehouse.

Most organizations that do not have a great deal of local autonomy and processing have a central data warehouse, as shown in Figure 6.4.

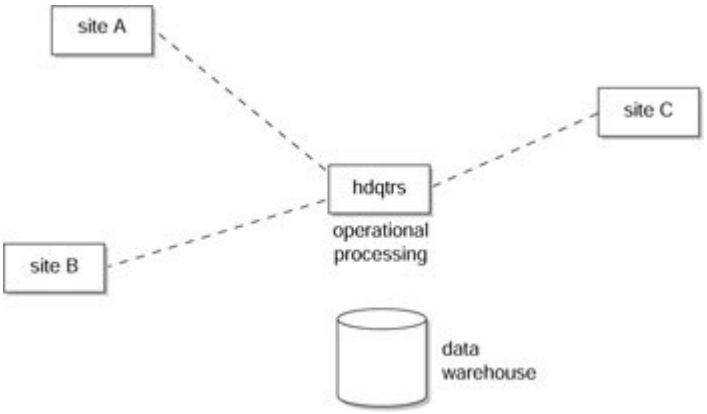

Figure 6.4: Most organizations have a centrally controlled, centrally housed data warehouse.

The Local Data Warehouse

A form of data warehouse, known as a local data warehouse, contains data that is of interest only to the local level. There might be a local data warehouse for Brazil, one for France, and one for Hong Kong. Or there might be a local data warehouse for car parts, motorcycles, and heavy trucks. Each local data warehouse has its own technology, its own data, its own processor, and so forth. Figure 6.5 shows a simple example of a series of local data warehouses.

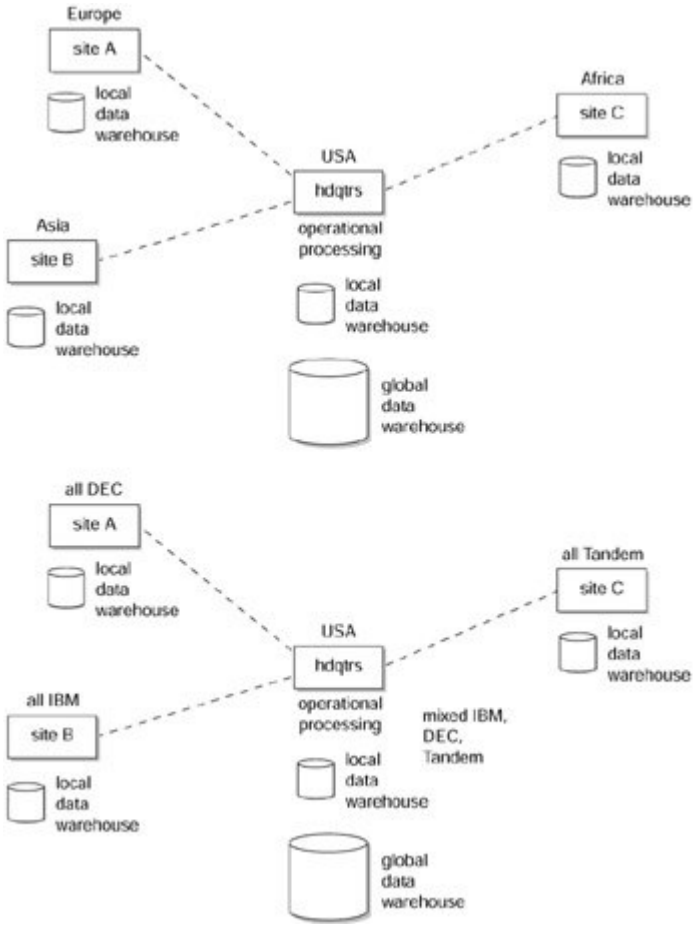

Figure 6.5: Some circumstances in which you might want to create a two-tiered level of data warehouse.

In Figure 6.5, a local data warehouse exists for different geographical regions or for different technical communities. The local data warehouse serves the same function that any other data warehouse serves, except that the scope of the data warehouse is local. For example, the data warehouse for Brazil does not have any information about business activities in France. Or the data warehouse for car parts does not have any data about motorcycles. In other words, the local data warehouse contains data that is historical in nature and is integrated within the local site. There is no coordination of data or structure of data from one local data warehouse to another.

The Global Data Warehouse

Of course, there can also be a global data warehouse, as shown in Figure 6.6. The global data warehouse has as its scope the corporation or the enterprise, while each of the local

data warehouses within the corporation has as its scope the local site that it serves. For example, the data warehouse in Brazil does not coordinate or share data with the data warehouse in France, but the local data warehouse in Brazil does share data with the corporate headquarters data warehouse in Chicago. Or the local data warehouse for car parts does not share data with the local data warehouse for motorcycles, but it does share data with the corporate data warehouse in Detroit. The scope of the global data warehouse is the business that is integrated across the corporation. In some cases, there is considerable corporate integrated data; in other cases, there is very little. The global data warehouse contains historical data, as do the local data warehouses. The source of the data for the local data warehouses is shown in Figure 6.7, where we see that each local data warehouse is fed by its own operational systems. The source of data for the corporate global data warehouse is the local data warehouses, or in some cases, a direct update can go into the global data warehouse.

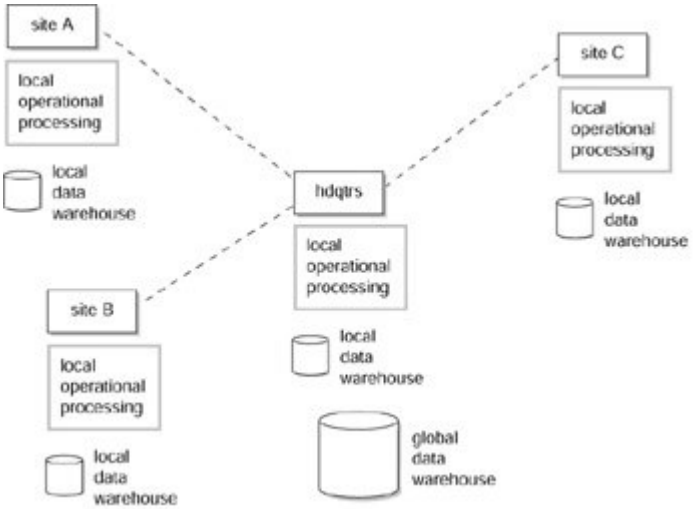

Figure 6.6: What a typical distributed data warehouse might look like.

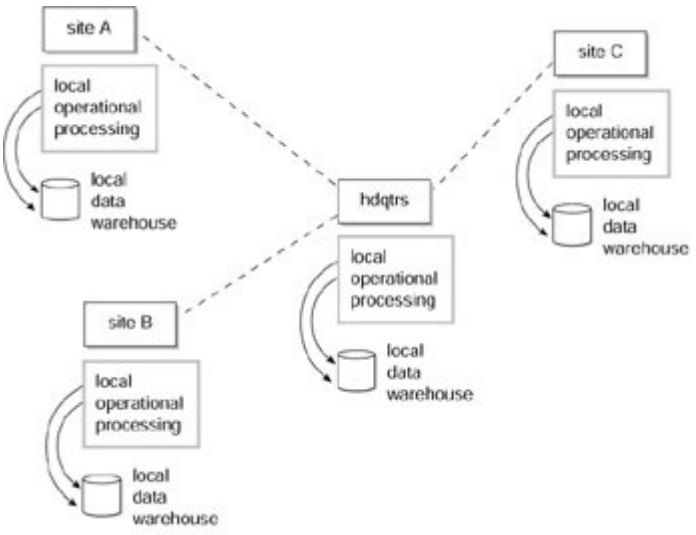

Figure 6.7: The flow of data from the local operational environment to the local data warehouse.

The global data warehouse contains information that must be integrated at the corporate level. In many cases, this consists only of financial information. In other cases, this may mean integration of customer information, product information, and so on. While a considerable amount of information will be peculiar to and useful to only the local level, other corporate common information will need to be shared and managed corporately. The global data warehouse contains the data that needs to be managed globally.

An interesting issue is commonality of data among the different local data warehouses. Figure 6.8 shows that each local warehouse has its own unique structure and content of data. In Brazil there may be much information about the transport of goods up and down the Amazon. This information is of no use in Hong Kong and France. Conversely, information might be stored in the French data warehouse about the trade unions in France and about trade under the Euro that is of little interest in Hong Kong or Brazil.

Or in the case of the car parts data warehouse, an interest might be shared in spark plugs among the car parts, motorcycle, and heavy trucks data warehouses, but the tires used by the motorcycle division are not of interest to the heavy trucks or the car parts division. There is then both commonality and uniqueness among local data warehouses.

Any intersection or commonality of data from one local data warehouse to another is purely coincidental. There is no coordination whatsoever of data, processing structure, or definition between the local data warehouses shown in Figure 6.8.

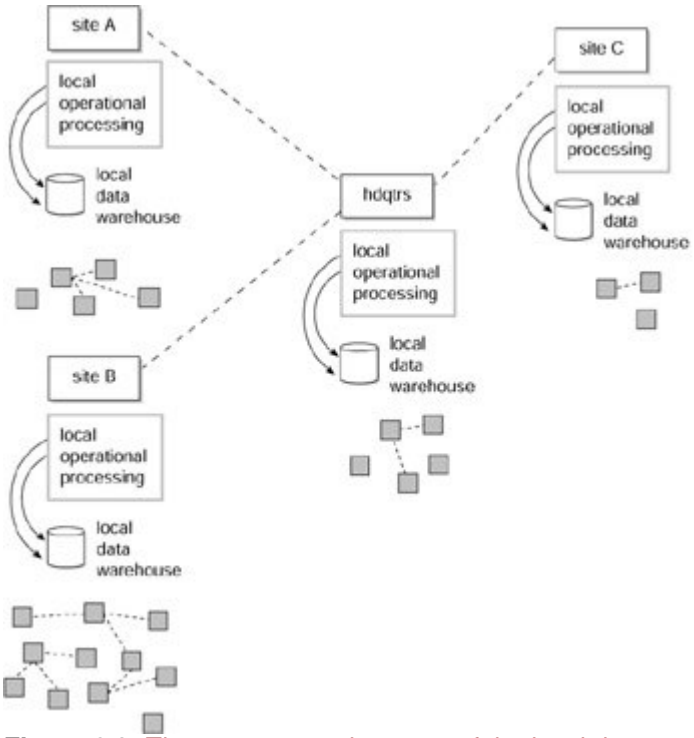

Figure 6.8: The structure and content of the local data warehouses are very different.

However, it is reasonable to assume that a corporation will have at least some natural intersections of data from one local site to another. If such an intersection exists, it is best contained in a global data warehouse. Figure 6.9 shows that the global data warehouse is fed from existing local operational systems. The common data may be financial information, customer information, parts vendors, and so forth.

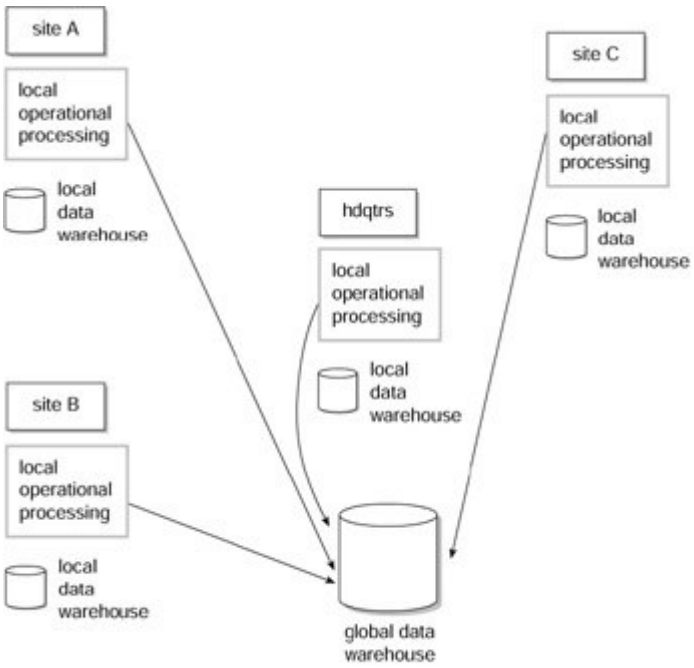

Figure 6.9: The global data warehouse is fed by the outlying operational systems.

Intersection of Global and Local Data

Figure 6.9 shows that data is being fed from the local data warehouse environment to the global data warehouse environment. The data may be carried in both warehouses, and a simple transformation of data may occur as the data is placed in the global data warehouse. For example, one local data warehouse may carry its information in the Hong Kong dollar but convert to the U.S. dollar on entering the global data warehouse. Or the French data warehouse may carry parts specifications in metric in the French data warehouse but convert metric to English measurements on entering the global data warehouse.

The global data warehouse contains data that is common across the corporation and data that is integrated. Central to the success and usability of the distributed data warehouse environment is the mapping of data from the local operational systems to the data structure of the global data warehouse, as seen in Figure 6.10. This mapping determines which data goes into the global data warehouse, the structure of the data, and any conversions that must be done. The mapping is the most important part of the design of the global data warehouse, and it will be different for each local data warehouse. For instance, the way that the Hong Kong data maps into the global data warehouse is different from how the Brazil data maps into the global data warehouse, which is yet different from how the French map their data into the global data warehouse. It is in the mapping to the global data warehouse that the differences in local business practices are accounted for.

The mapping of local data into global data is easily the most difficult aspect of building the global data warehouse.

Figure 6.10 shows that for some types of data there is a common structure of data for the global data warehouse. The common data structure encompasses and defines all common data across the corporation, but there is a different mapping of data from each local site into the global data warehouse. In other words, the global data warehouse is designed and defined centrally based on the definition and identification of common corporate data, but the

mapping of the data from existing local operational systems is a choice made by the local designer and developer.

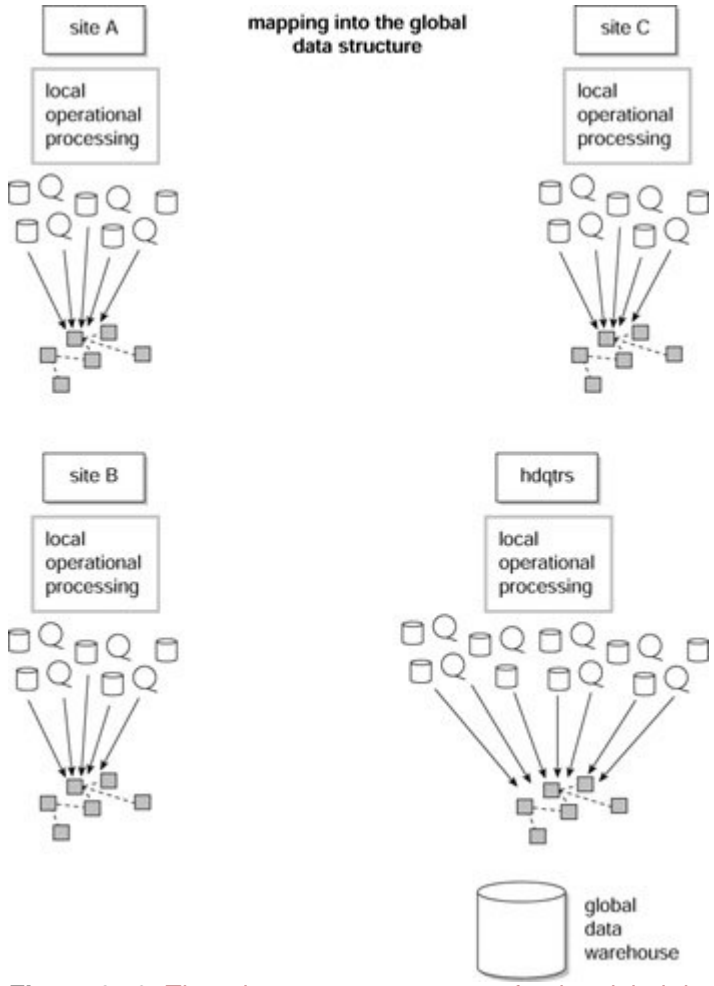

Figure 6.10: There is a common structure for the global data warehouse. Each local site maps into the common structure differently.

It is entirely likely that the mapping from local operational systems into global data warehouse systems will not be done as precisely as possible the first time. Over time, as feedback from the user is accumulated, the mapping at the local level improves. If ever there were a case for iterative development of a data warehouse, it is in the creation and solidification of global data based on the local mapping.

A variation of the local/global data warehouse structure that has been discussed is to allow a global data warehouse "staging" area to be kept at the local level. Figure 6.11 shows that each local area stages global warehouse data before passing the data to the central location. For example, say that in France are two data warehouses—one a local data warehouse used for French decisions. In this data warehouse all transactions are stored in the French franc. In addition, there is a "staging area" in France, where transactions are stored in U.S. dollars. The French are free to use either their own local data warehouse or the staging area for decisions. In many circumstances, this approach may be technologically mandatory. An important issue is associated with this approach: Should the locally staged global data warehouse be emptied after the data that is staged inside of it is transferred to the global level? If the data is not deleted locally, redundant data will exist. Under certain

conditions, some amount of redundancy may be called for. This issue must be decided and policies and procedures put into place.

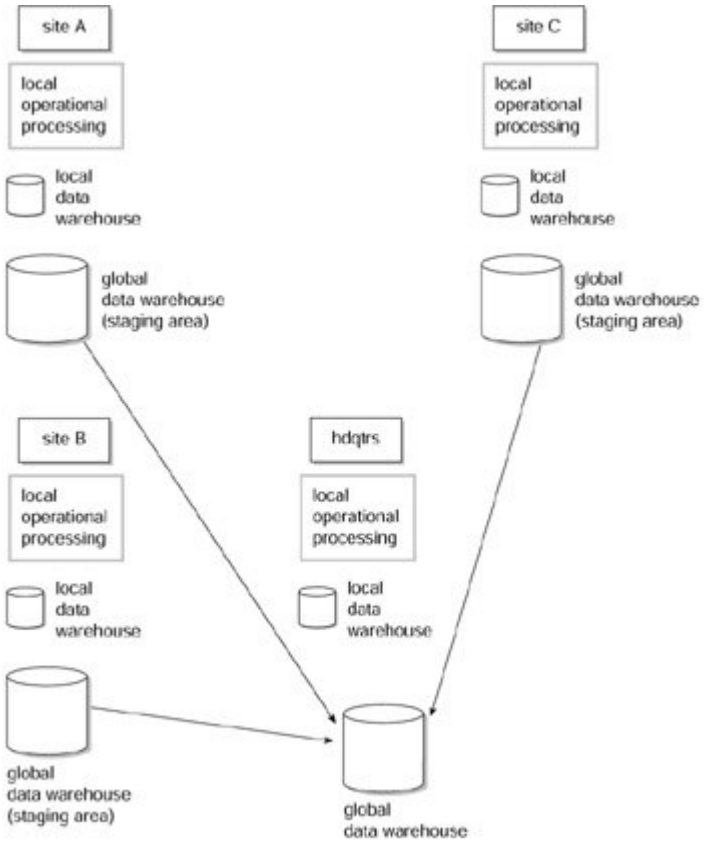

Figure 6.11: The global data warehouse may be staged at the local level, then passed to the global data warehouse at the headquarters level.

For example, the Brazilian data warehouse may create a staging area for its data based on American dollars and the product descriptions that are used globally. In the background the Brazilians may have their own data warehouse in Brazilian currency and the product descriptions as they are known in Brazil. The Brazilians may use both their own data warehouse and the staged data warehouse for reporting and analysis.

Though any of several subject areas may be candidates for the first global data warehouse development effort, many corporations begin with corporate finance. Finance is a good starting point for the following reasons:

- It is relatively stable.
- It enjoys high visibility.
- It is only a fraction of the business of the organization (except, of course, for organizations whose business is finance).
- It is always at the nerve center of the organization.
- It entails only a modicum of data.

In the case of the global warehouse being discussed, the Brazilian, the French, and the Hong Kong data warehouses would all participate in the building of a corporatewide financial data warehouse. There would be lots of other data in the operations of the Brazilian, French, and Hong Kong business units, but only the financial information would flow into the global data warehouse.

Because the global data warehouse does not fit the classical structure of a data warehouse as far as the levels of data are concerned, when building the global data warehouse, one must recognize that there will be some anomalies. One such anomaly is that the detailed data (or, at least, the source of the detailed data) resides at the local level, while the lightly summarized data resides at the centralized global level. For example, suppose that the headquarters of a company is in New York and it has outlying offices in Texas, California, and Illinois. The details of sales and finance are managed independently and at a detailed level in Texas, California, and Illinois. The data model is passed to the outlying regions, and the needed corporate data is translated into the form that is necessary to achieve integration across the corporation. Upon having made the translation at the local level, the data is transmitted to New York. The raw, untranslated detail still resides at the local level. Only the transformed, lightly summarized data is passed to headquarters. This is a variation on the theme of the classical data warehouse structure.

Redundancy

One of the issues of a global data warehouse and its supporting local data warehouses is redundancy or overlap of data. Figure 6.12 shows that, as a policy, only minimal redundant data exists between the local levels and the global levels of data (and in this regard, it matters not whether global data is stored locally in a staging area or locally). On occasion, some detailed data will pass through to the global data warehouse untouched by any transformation or conversion. In this case, a small overlap of data from the global data warehouse to the local data warehouse will occur. For example, suppose a transaction occurs in France for US$10,000. That transaction may pass through to the global data warehouse untouched.

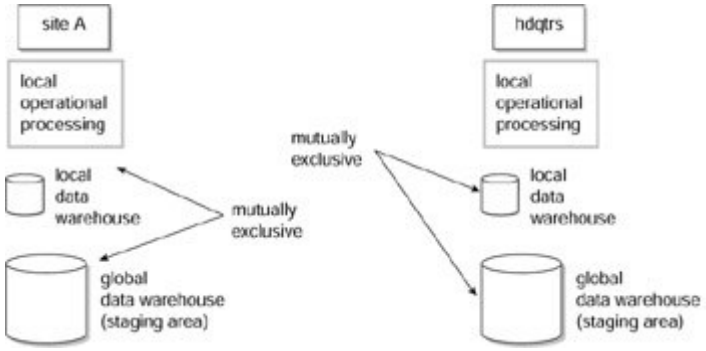

Figure 6.12: Data can exist in either the local data warehouse or the global data warehouse, but not both.

On the other hand, most data passes through some form of conversion, transformation, reclassification, or summarization as it passes from the local data warehouse to the data warehouse. In this case, there is—strictly speaking—no redundancy of data between the two environments. For example, suppose that a HK$175,000 transaction is recorded in Hong Kong. The transaction may be broken apart into several smaller transactions, the dollar amount may be converted, the transaction may be combined with other transactions, and so forth. In this case, there is certainly a relationship between the detailed data found in the local data warehouse and the data found in the global data warehouse. But there is no redundancy of data between the two environments.

A massive amount of redundancy of data between the local and the global data warehouse environments indicates that the scopes of the different warehouses probably have not been defined properly. When massive redundancy of data exists between the local and the global data warehouse environments, it is only a matter of time before spider web systems start to appear. With the appearance of such systems come many problems—reconciliation of

inconsistent results, inability to create new systems easily, costs of operation, and so forth. For this reason, it should be a matter of policy that global data and local data be mutually exclusive with the exception of very small amounts of data that incidentally overlap between the two environments.

Access of Local and Global Data

In line with policies required to manage and structure the local and the global data warehouses is the issue of access of data. At first, this issue seems to be almost trivial. The obvious policy is that anyone should be able to get at any data. Some important ramifications and nuances come into play, however.

Figure 6.13 shows that some local sites are accessing global data. Depending on what is being asked for, this may or may not be an appropriate use of data warehouse data. For example, an analyst in Brazil may be analyzing how Brazilian revenues compare to total corporate revenues. Or a person in France may be looking at total corporate profitability. If the intent of the local analysis is to improve local business, the access of global data at the local level is probably a good policy. If the global data is being used informationally, on a one-time-only basis, and to improve local business practices, the access of global data at the local level is probably acceptable.

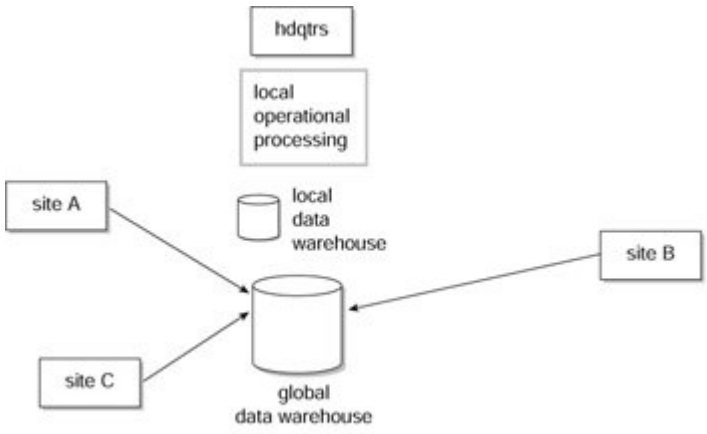

Figure 6.13: An important issue to be resolved is whether local sites should be accessing the global data warehouse.

As a principle, local data should be used locally and global data should be used globally. The question must be raised, then, why is global analysis being done locally? For example, suppose a person in Hong Kong is comparing total corporate profitability with that of another corporation. There is nothing wrong per se with this analysis, except that this sort of global analysis is best performed at the headquarters level. The question must be asked—what if the person in Hong Kong finds that globally the corporation is not competing well with other corporations? What is the person in Hong Kong going to do with that information? The person may have input into global decisions, but he or she is not in a position to effect a such a decision. Therefore, it is questionable whether a local business analyst should be looking at global data for any other purpose than that of improving local business practices. As a rule, the local business analyst should be satisfied using local data.

Another issue is the routing of requests for information into the architected information environment. When only a single central data warehouse existed, the source of a request for information was not much of an issue. But when data is distributed over a complex

landscape, such as a distributed data warehouse landscape, as shown in Figure 6.14, there is the consideration of ensuring the request originated from the appropriate place.

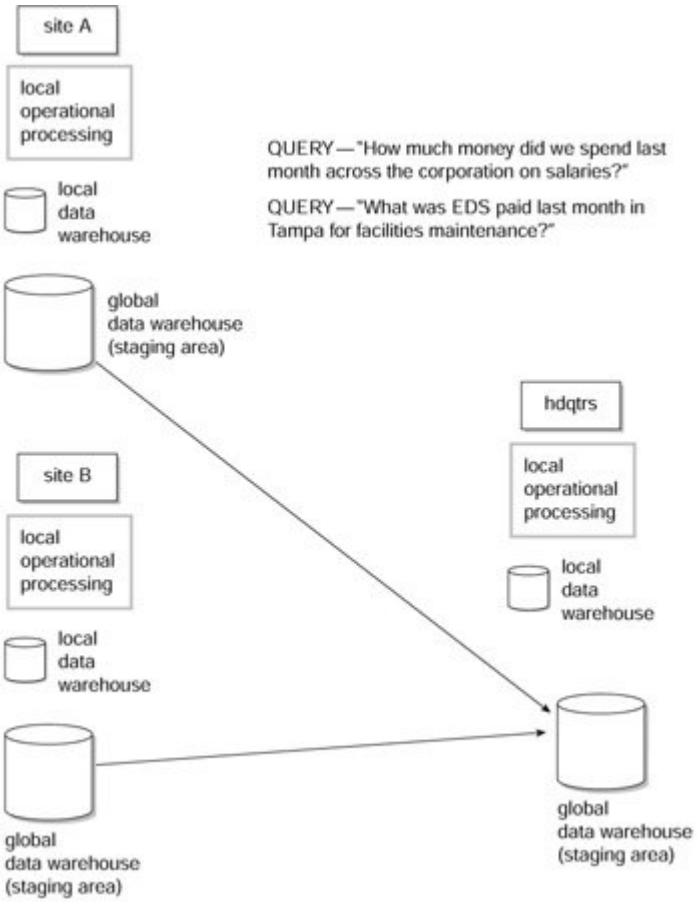

Figure 6.14: Queries need to be routed to different parts of the architecture to be answered properly.

For example, asking a local site to determine total corporate salaries is inappropriate, as is asking the central data warehouse group what a contractor was paid last month at a particular site for a particular service. With local and global data there is the issue of origin of request, something not encountered in the simple centralized data warehouse environment.

Another important issue of local/global distributed data warehousing is the transfer of data from the local data warehouse to the global data warehouse. There are many facets to this issue:

- How frequently will the transfer of data from the local environment to the global environment be made? Daily? Weekly? Monthly? The rate of transfer depends on a combination of factors. How quickly is the data needed in the global data warehouse? How much activity has occurred at the local level? What volume of data is being transported?
- Is the transportation of the data from the local environment to the global data warehouse across national lines legal? Some countries have Draconian laws that prevent the movement of certain kinds of data across their national boundaries.
- What network will be used to transport the data from the local environment to the global environment? Is the Internet safe enough? Reliable enough? Can the Internet

safely transport enough data? What is the backup strategy? What safeguards are in place to determine if all of the data has been passed?

- What safeguards are in place to determine whether data is being hacked during transport from the local environment to the global environment?
- What window of processing is open for transport of data from the local environment to the global environment? Will transportation have to be done during the hours when processing against the data warehouse will be heavy?
- What technology is the local data in, and what technology is the global data in? What measures must be taken to convert the local technology to the global technology? Will data be lost in the translation?

Many issues relate to the physical movement of the data into the global data warehouse environment. In some cases, these issues are mundane; in others, they are anything but.

A related yet separate issue not addressed in this chapter is global operational data. Thus far, this chapter has assumed that every local site has its own unique operational data and processing. However, it is entirely possible that there is a degree of commonality between the operational systems found at each local site. In this case, some degree of corporate operational data and processing might be desirable. For example, some customers may need to be handled globally, such as large multinational corporations like Coca-Cola, McDonalds, IBM, and AT&T. Pricing considerations, size-of-order considerations, and delivery considerations may be treated differently globally than similar decisions are treated locally. In the case of global operational processing, that global operational data merely becomes another source of data into the global data warehouse. But there is still the distinction between operational data and DSS informational data.

Underlying the whole issue of the distributed data warehouse is complexity. In a simple central data warehouse environment, roles and responsibilities are fairly straightforward. In a distributed data warehouse environment, however, the issues of scope, coordination, meta data, responsibilities, transfer of data, local mapping, and more make the environment complex indeed.

One of the major considerations for a global data warehouse is whether the data warehouse should be built centrally or globally. While tempting to say that a global data warehouse should be designed and built centrally, doing so is patently a mistake. With a centralized construction of a global data warehouse, there is—at best—only a marginal local buy-in to the global data warehouse. This means that the definition of the mapping between the local systems and the needs for global data is done centrally, not locally. To succeed, there must be local management and control of the mapping process. Stated differently, the single most difficult part of the building and population of the global data warehouse is the mapping of the local data to the global data. And this mapping cannot be done centrally; it must be done locally.

For example, suppose the headquarters organization tries to map Brazilian data into the global data warehouse. This poses the following problems:

- Portuguese is not the native tongue of the headquarters organization.
- Headquarters personnel do not understand local business practices and customs.
- Headquarters personnel do not understand local legacy applications.
- Headquarters personnel do not understand the local data warehouse.
- Headquarters personnel are not on hand to keep abreast of day-to-day changes in local systems.

There are then a plethora of reasons why the mapping of local data into the global data warehouse environment cannot be done by centralized personnel.

Therefore, the local organization must be a part of the building of the global data warehouse.

A final observation is that the local data needs to be in as flexible a form as possible. This usually means that the local data must be organized relationally. If the local data is organized in a star join multidimensional format, it will be difficult to break apart and restructure for the global data warehouse.

The Technologically Distributed Data Warehouse

The spread of a company over multiple geographical locations or over many different product lines is not the only reason why a distributed data warehouse is appealing. There are other rationales as well. One case for a different type of a distributed warehouse is that of placing a data warehouse on the distributed technology of a vendor. Client/server technology fits this requirement nicely.

The first question is, can a data warehouse be placed on distributed technology? The answer is yes. The next question is, what are the advantages and disadvantages of using distributed technology for a data warehouse? The first advantage of a technologically distributed data warehouse is that the entry cost is cheap. In other words, the cost of hardware and software for a data warehouse when initially loaded onto distributed technology is much less than if the data warehouse were initially loaded onto classical, large, centralized hardware. The second advantage is that there is no theoretical limit to how much data can be placed in the data warehouse. If the volume of data inside the warehouse begins to exceed the limit of a single distributed processor, then another processor can be added to the network, and the progression of adding data continues in an unimpeded fashion. Whenever the data grows too large, another processor is added.

Figure 6.15 depicts a world in which there may be an infinite amount of data in the data warehouse. This is appealing because a data warehouse will contain much data (but not an infinite amount!).

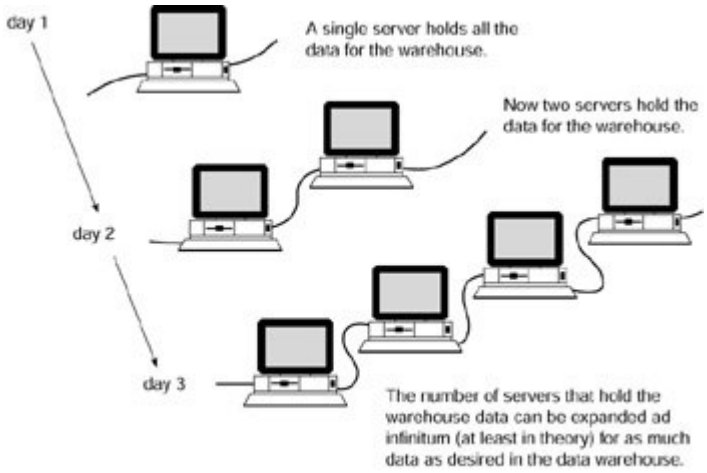

Figure 6.15: The progression of adding more servers to hold the data warehouse data.

There are, however, some considerations. As the data warehouse starts to expand beyond a few processors (i.e., servers), an excessive amount of traffic starts to appear on the network. The increased traffic appears when a request for data overlaps from one processor to another. For example, suppose one processor holds data for the year 1998, another processor for 1999, another for 2000, and yet another for 2001. When a request is made for

data from 1998 to 2001, the result set for that query must pass over the boundaries of the processor that originally held the data. In this case, data from four processors must be gathered. In the process, data passes across the network and increases the traffic.

The issue arises not only of a query accessing data housed by multiple processors, but of a large amount of data needing to be transported from a single processor. For example, suppose a query wishes to access all 1999 data and all 2000 data. Such a query will pull data from one or the other processor. Figure 6.16 depicts a query that wishes to access a large amount of data from multiple servers.

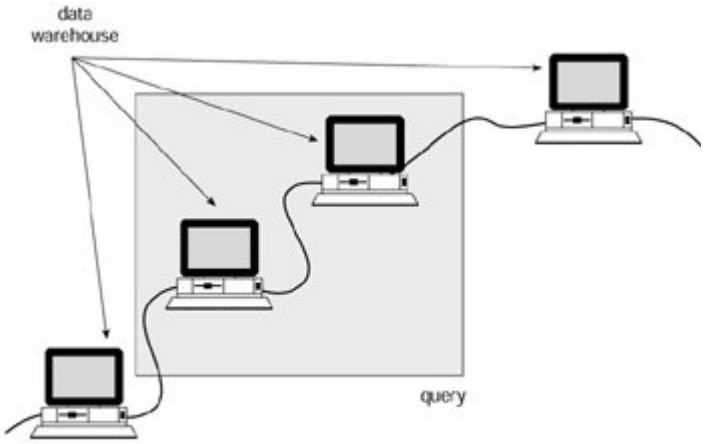

Figure 6.16: A query that accesses a large amount of data from multiple data warehouse servers.

Techniques and approaches do of course exist to deal with a data warehouse that is technically distributed over multiple servers. Ironically, the problems grow with time, as the warehouse becomes fully populated and as the number of servers grows. In the early days of a distributed data warehouse when there is very little data and only a few servers, the problems discussed here are hardly obvious. In the more mature days of a data warehouse, however, the data and the processing environment become more difficult to manage.

The Independently Evolving Distributed Data Warehouse

Yet a third flavor of distributed data warehouse is one in which independent data warehouses are developed concurrently and in an uncontrolled manner.

The first step many corporations take in data warehousing is to put up a data warehouse for a financial or marketing organization. Once success follows the initial warehouse effort, other parts of the corporation naturally want to build on the successful first effort. In short order, the data warehouse architect has to manage and coordinate multiple data warehouse efforts within the organization.

The Nature of the Development Efforts

The first issue facing the data architect who must manage multiple data warehouse development efforts is the nature of the development efforts. Unless the data architect knows what kinds of efforts are occurring and how they relate to the overall architecture, he or she will have a very difficult time managing and coordinating them. Because the issues of development vary considerably depending on the approach, different types of data warehouse development efforts require very different approaches to management.

Multiple data warehouse development efforts occur in four typical cases, as outlined in Figure 6.17.

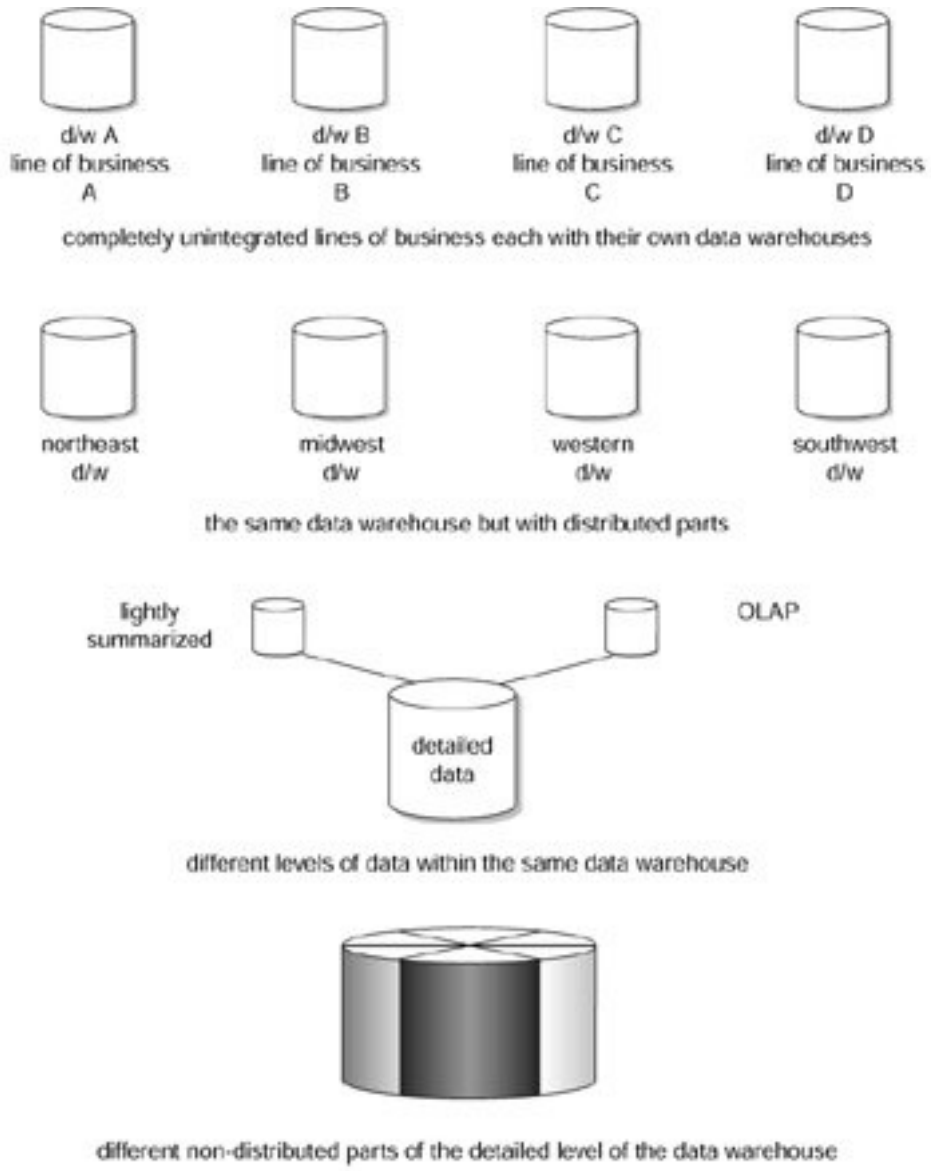

Figure 6.17: Four possible meanings to "multiple groups building the data warehouse," each interpretation very different from the others.

In the first, rare case shown in Figure 6.17, a corporation has totally separate and unintegrated lines of business for which data warehouses are being independently built by different development groups. The diverse lines of business report to the corporation, but other than sharing a company name, there is no business integration or sharing of data within the company. Such a corporate

structure is not unknown, but it is not common. In this case, there is very little danger that one data warehouse development effort will conflict with another. Accordingly, there is little or no need for cross-management and coordination of data warehouse development efforts.

The second case of multiple data warehouse efforts occurring simultaneously happens when a corporate distributed data warehouse is being built and various development groups are creating different parts of the same data warehouse. In this case, the same detailed data is being built by different groups of developers, but the data is distributed across different locations. For example, suppose a car manufacturer is building a manufacturing data warehouse in Detroit and another manufacturing data warehouse in Canada. The same detailed data finds its way into the data warehouse in both places. Unless extraordinary measures are taken, likely many conflicting analyses will occur. This case for multiple development efforts is as common as the previous case is rare. Yet because this case is so common, it requires a great deal of attention. It requires discipline and close coordination among the groups to achieve a collectively satisfying result. The danger of not coordinating this effort is that much waste may occur through redundant storage and manipulation of massive amounts of data. If data is created redundantly, the resulting data warehouse may well be ineffective because there will be a classic spider web in the DSS environment.

The third case of multiple data warehouse development occurring simultaneously happens when different groups are building different levels of data (i.e., summarized data and detailed data) in the data warehouse environment. Like the preceding case, this scenario is also very common. However, for a variety of reasons, it is much easier to manage than either of the two previous cases. Because of the differences in the levels of data, there are different uses and expectations. And coordination between the two groups is likely to be a straightforward exercise. For example, one group of people might be building a data warehouse to capture and analyze each bank transaction at the lowest level of detail. Another group of analysts might be creating customer records summarized up to the monthly level. The interface between the two groups is simple. The detailed transactions are summarized on a monthly basis to create the aggregate/summary record for the customer.

The fourth case occurs when multiple groups are trying to build different parts of the current level of detail of the data warehouse environment in a nondistributed manner. This is a somewhat rare phenomenon, but when it occurs, it mandates special attention. There is much at stake in this last case, and the data architect must be aware of what the issues are and how they relate to success.

Each of these cases are discussed in the following sections, along with their issues, advantages, and disadvantages.

Completely Unrelated Warehouses

The building and operation of completely unrelated warehouses is shown in Figure 6.18. A corporation has four lines of business—golf course management, a steel mill, retail banking, and a fast-food franchise. There is no integration of

Figure 6.18: Four totally independent enterprises where there is little or no integration of data at the business level.

the businesses whatsoever. A customer of one business may be a customer of another business, and there is no desire to link the two relationships. The ongoing data warehouse efforts have no reason to be coordinated. All the way from modeling to selection of base

technology (i.e., platform, DBMS, access tools, development tools, etc.), each of the different businesses could operate as if it were completely autonomous.

For all the autonomy of the lines of business, they are necessarily integrated at one level: the financial balance sheet. If the different lines of business report to a single financial entity, there must be integration at the balance-sheet level. In this situation, a corporate data warehouse might need to be built that reflects the corporate finances. Figure 6.19 shows a corporate financial data warehouse sitting above the different businesses.

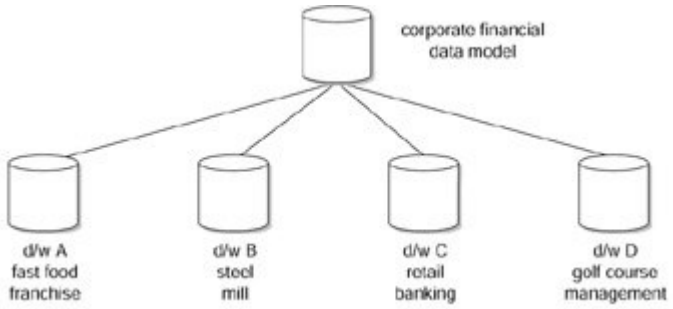

Figure 6.19: Even the most disparate of business enterprises share common corporate financial data.

The financial corporate data warehouse contains simple (and abstract) entities such as expenses, revenues, capital expenditures, depreciation, and the like. There is very little, if any, business data beyond that found on every balance sheet. (In other words, no attempt is made for a common corporate description of customer, product, sale, and so on in the financial data warehouse.) Of course, the data feeding the corporate financial data warehouse depicted in Figure 6.19 may come either from the "local" data warehouse or from the operational systems found at the individual operating company level.

Meta data is vital at the local level. With a corporate financial data warehouse, it is also needed at the corporate financial level. However, in this case, because no real business integration exists, there is no need to tie any of the meta data together.

Distributed Data Warehouse Development

Unlike the case of the unrelated warehouses, most businesses have some degree of integration among their disparate parts. Very few businesses are as autonomous as those depicted in Figure 6.19. A much more common scenario for the development of multiple data warehouse efforts is shown in Figure 6.20.

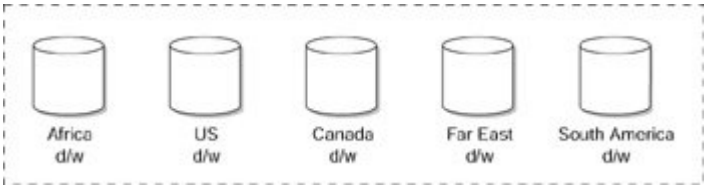

Figure 6.20: Logically the same data warehouse.

In Figure 6.20 a corporation has different sites in different parts of the world—one in the United States and Canada, one in South America, one in the Far East, and one in Africa. Each site has its own unique data, with no overlap of data—particularly of detailed transaction data—from one site to the next. The company wants to create a data warehouse

for each of the disparate entities as a first effort in achieving an architected environment. There is some degree of business integration among the different organizations. At the same time, it is assumed that distinct business practices are carried on in each locale. Such an organization of corporate entities is common to many companies.

The first step many organizations make toward data warehousing is to create a local data warehouse at each geographical entity. Figure 6.21 shows the creation of a local data warehouse.

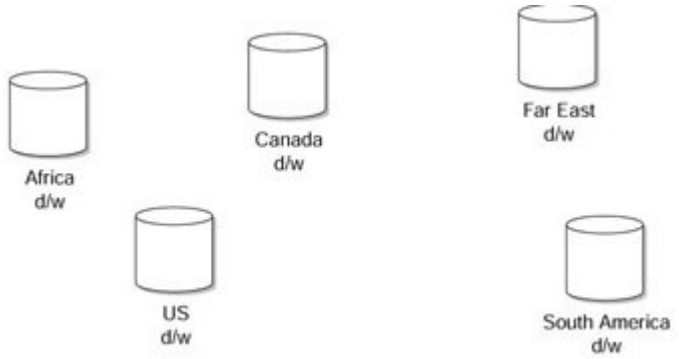

Figure 6.21: Local data warehouses are built at each of the autonomous operating divisions.

Each locale builds its own unique autonomous data warehouse according to its needs. Note that there is no redundant detailed data among the different locales, at least as far as transaction data is concerned. In other words, a unit of data reflecting transactions that belongs in Africa would never be found in the local data warehouse for Europe.

There are several pros and cons to this approach to building the distributed corporate data warehouse. One advantage is that it is quick to accomplish. Each local group has control over its design and resources. A sense of autonomy and control may make each local organization happy. As such, the benefits of the data warehouse can be proven throughout the corporation on a real-time basis. Within six months the local data warehouses can be up and running, and the organization at the local level can be deriving the benefits. The disadvantage is that if there is any commonality in the structure (not the content) of data across the organization, this approach does nothing to recognize or rationalize that commonality.

Coordinating Development across Distributed Locations

An alternative approach is to try to coordinate the local data warehouse development efforts across the different local organizations. This sounds good in theory, but in execution it has not proved to be effective. The local development groups never collectively move at the same pace, and the local groups look on the central development group trying to coordinate the many local development efforts as a hindrance to progress. A separate data model is built to provide the foundation of the data warehouse design for each of the separate locales.

One day, after the worth of data warehousing has been proven at the local level, the corporation decides to build a corporate data warehouse (see Figure 6.22).

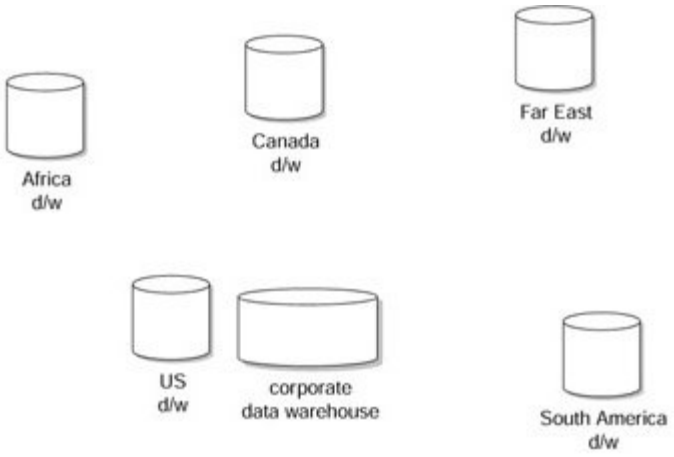

Figure 6.22: One day the decision is made to build a corporate data warehouse.

The corporate data warehouse will reflect the business integration across the different divisions and locales. The corporate data warehouse will be related to, but still distinct from, the local data warehouses. The first step in building the corporate data warehouse is to create a corporate data model for that portion of the business that will be reflected in the corporate data warehouse. As a general rule, the corporate data model that is built for the first iteration of the corporate data warehouse will be small and simple, and it will be limited to a subset of the business. Figure 6.23 illustrates the building of the corporate data model after which the corporate data warehouse will be shaped.

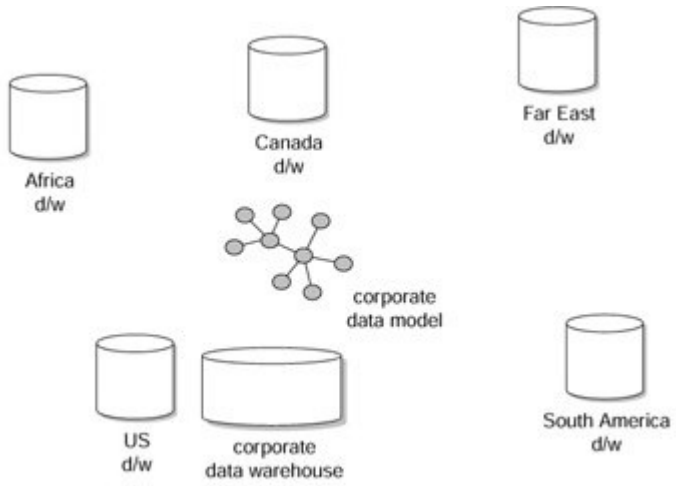

Figure 6.23: The corporate data model is created.

The Corporate Data Model—Distributed

The corporate data model reflects the integration of business at the corporate level. As such, the corporate data model may overlap considerably with portions of the local data models. Such an overlap is healthy and normal. In other cases, the corporate data model will be different from the local data models. In any case, it is up to the local organization to determine how the fit is to be made between the corporate need for data and the local ability to provide it. The local organization knows its own data better than anyone, and it is best equipped to show how local data needs to be shaped and reshaped to meet the specifications of the corporate design of data for the data warehouse.

While there may very well be overlap in the structural design of data from one local level to the next, there is no overlap to any great extent in the content of data. Figure 6.24 shows the building and population of the corporate data warehouse from the local levels.

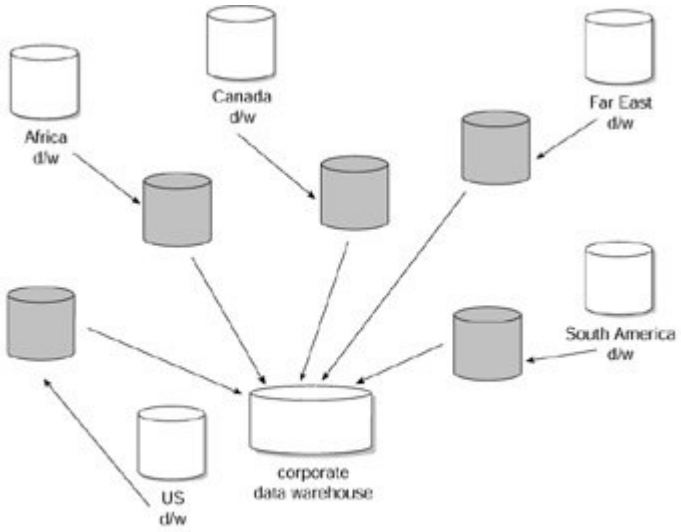

Figure 6.24: The corporate data warehouse is loaded from the different autonomous operating companies.

The source of the data going to the corporate data warehouse can come from the local data warehouse or from the local operational systems. The determination of the system of record should be a decision that is made entirely at the local level. Most certainly, several iterations of definition of the system of record will be needed.

In addition, an important design issue is how to create and transport the local system of record data from the technology found at the local level into the technology of the corporate data warehouse. In some cases, the official "staged" data is kept at the local level. In other cases, the staged data is passed on to the corporate environment with no access at the local level.

As a rule, the data in the corporate data warehouse is simple in structure and concept. Figure 6.25 shows that data in the corporate data warehouse appears to be detailed data to the DSS analyst at the corporate level, and at the same time it appears tobe summary data to the DSS analyst at the local level. This apparent contradiction is reconciled by the fact that the appearance of summarization or detail is strictly in the eye of the beholder.

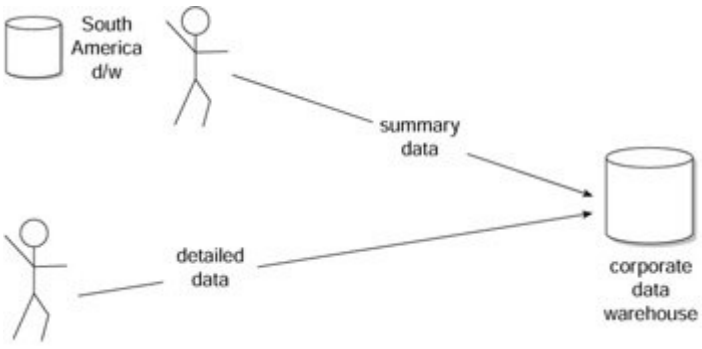

Figure 6.25: What is summary at one level is detailed at another.

The corporate data warehouse of the distributed database can be contrasted with the corporate financial data warehouse of the completely unrelated companies. Figure 6.26 makes this comparison.

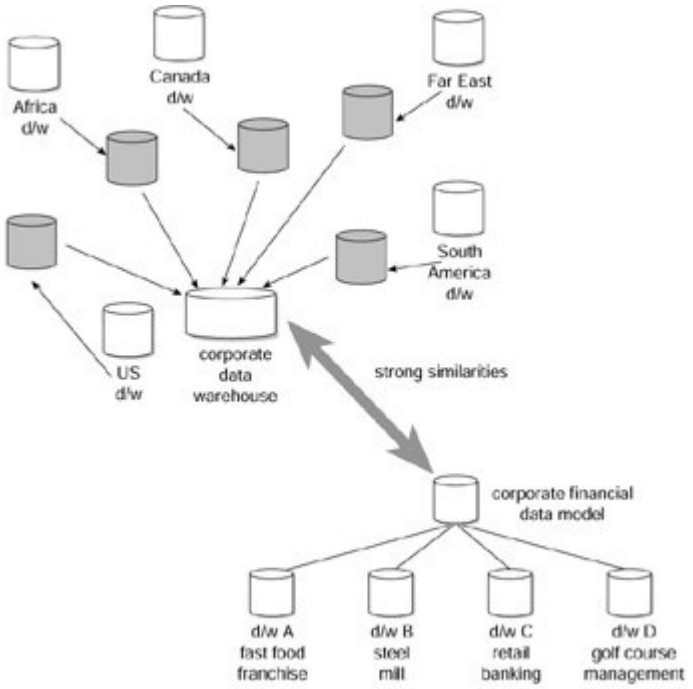

Figure 6.26: The data warehouse of a distributed corporation can be very similar to that of unrelated companies.

In many ways, the data warehouse of the distributed corporation is very similar to the data warehouse of the unrelated companies. However, although there are similarities in design and operation, there is one major difference. The corporate distributed data warehouse extends into the business itself, reflecting the integration of customer, vendor, product, and so forth. As such, the corporate distributed data warehouse represents the very fabric of the business itself. The corporate data warehouse for unrelated companies, though, is for finance alone. The instant that there is a desire to use the corporate financial data warehouse for anything other than the financial relationship of the different parts of the corporation, there will be disappointment with the corporate financial data warehouse. The difference between the two data warehouses, then, is one of depth.

Meta Data in the Distributed Warehouse

Meta data plays a very important role across the distributed corporate data warehouse. It is through meta data that the coordination of the structure of data is achieved across the many different locations where the data warehouse is found. Not surprisingly, meta data provides the vehicle for the achievement of uniformity and consistency.

Building the Warehouse on Multiple Levels

The third scenario of a company's simultaneous data warehouse development occurs when different development groups are building different levels of the data warehouse, as seen in Figure 6.27. This case is very different from the case of the distributed data warehouse development. In this case, Group A is building the high level of summarization of data,

Group B is building the middle level of summarization, and Group C is building the current level of detail.

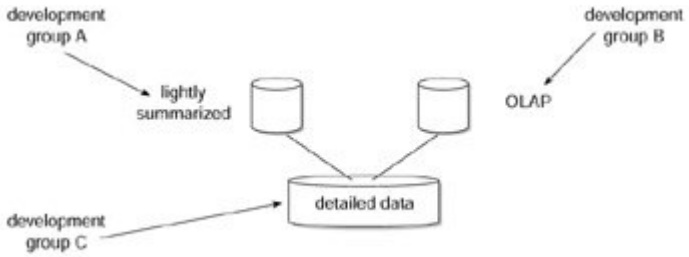

Figure 6.27: Different development groups are developing different parts of the data warehouse environment at different levels of the architecture.

The scenario of multiple levels of data warehouse development is very common. Fortunately, it is the easiest scenario to manage, with the fewest risks.

The primary concern of the data architect is to coordinate the efforts of the different development teams, both in terms of the specification of content and structure and in terms of the timing of development. For example, if Group A is significantly ahead of Group B or C, there will be a problem: When Group A is ready to populate its databases at the summary level, there may be no detailed data to work with.

One of the interesting aspects of different groups building different levels of summarization of the same data warehouse is that it is the group that is building the current level of detail that uses the data warehouse data model. Figure 6.28 illustrates this relationship.

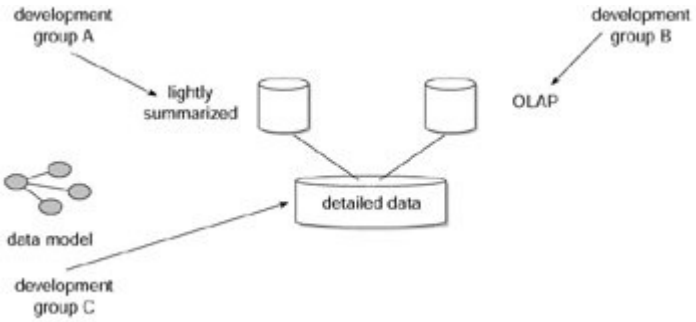

Figure 6.28: The development organization that is developing the lowest level of detail is the organization that uses the data model.

The data model for the data warehouse directly reflects the design and development effort by the group doing current-level detailed analysis and design. Of course, indirectly the data warehouse data model reflects the needs of all groups. But because other groups are summarizing from the data found at the current level of detail, they have their own interpretation of what is needed. In most cases, the groups working on the higher levels of summarization have their own data models that reflect their own specialized needs.

One of the issues of managing multiple groups building different levels of summarization is the technological platforms on which the data warehouse levels are built. Normally, different groups choose different technological platforms. In fact, for the different development groups to choose the same platform would be very unusual. There are several reasons for this, though the primary one is cost. The detailed level of data requires an industrial-strength platform because of the large volume of data that will have to be handled. The different

levels of summarization will require much less data, especially at the higher levels of summarization. It is overkill (and expensive) to place the higher levels of summarized data on the same platform as the detailed data (although it can be done).

Another reason is that the alternative platforms offer a wide variety of specialized software, much of which is not to be found on the monolithic platforms that house detailed data. In any case, whether the different levels of data are on a single platform or on multiple platforms, meta data must be carefully stored and managed, so that continuity from one level of detail to the next can be maintained.

Because different platforms are commonly used for the different levels of data that are being developed by different groups, the issue of interconnectivity arises. Figure 6.29 shows the need for interconnectivity from one level to the next.

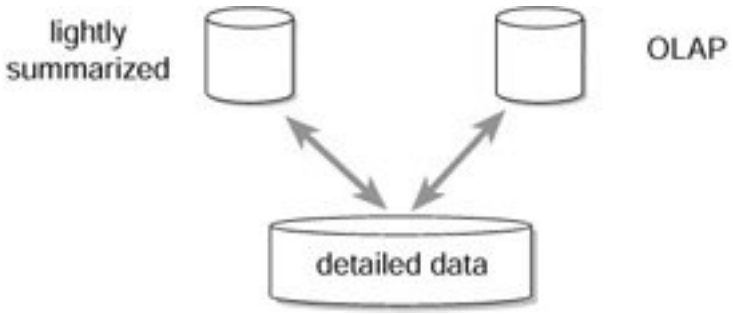

Figure 6.29: Interconnectivity between the different levels of the data warehouse is an important issue.

Several aspects of interconnectivity need to be addressed. One issue is the compatibility of access at the call level. In other words, is there compatible syntax between the two technologies that make up the detailed and the summary data between any two levels of the warehouse? If there is not at least some degree of compatibility, the interface will not be usable. Another aspect of the interconnectivity issue is the effective bandwidth of the interface. If very heavy traffic is being created by processing that occurs on either level of the data warehouse, the interface between the two environments can become a bottleneck.

However the groups that work on the data warehouse are coordinated, one requirement remains clear: The group that manages the lower-level detail must form a proper foundation of data for those groups that will be summarizing the data and creating a new level. This need is depicted in Figure 6.30.

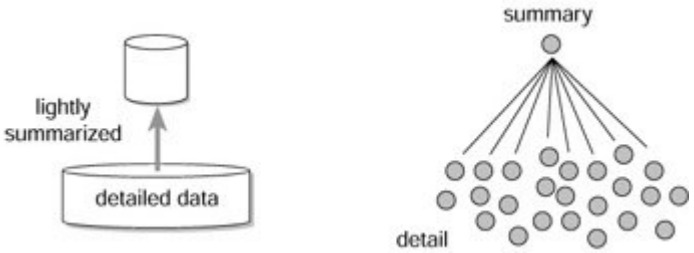

Figure 6.30: The detailed level forms the foundation for the summary level of data.

The coordination among the groups can be as simple as an agreement on a data model that satisfies the needs of all parties. Or the agreement can be much more elaborate, if circumstances warrant.

The coordination of the development efforts is another matter. There must be a time-sequenced arrangement among the different development groups so that no one group arrives at a point of needing data that has not yet been gathered at a lower level.

Multiple Groups Building the Current Level of Detail

An infrequent set of circumstances occurs when multiple development groups attempt to build the current level of detail in a data warehouse in a nondistributed manner. Figure 6.31 illustrates this phenomenon.

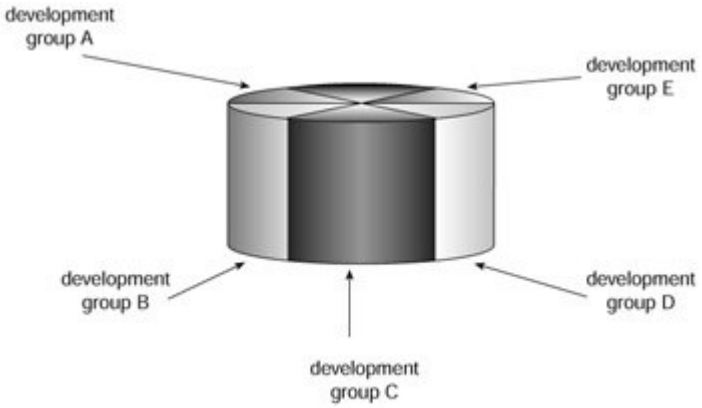

Figure 6.31: Different development groups that are developing the current level of detail for the data warehouse.

As long as the groups that are developing the current level of detail are developing mutually exclusive sets of data, there is little difficulty. In this case, if the development groups are working from a common data model and the different groups' platforms are compatible, no problems should ensue. Unfortunately, mutually exclusive data sets are the exception rather than the rule. It is much more common for the multiple development groups to be designing and populating some or all of the same data.

A series of problems arise when the groups overlap. The first problem is cost—in particular, the cost of storage and processing. The volumes of data that are found at the current detailed level are such that any amount of redundancy must be questioned, much less wholesale redundancy. The cost of processing the detail is likewise a major issue.

The second, more insidious issue is the introduction of the spider web into the DSS environment. With massive amounts of redundant detailed data, it is almost axiomatic that misinterpretation of data caused by redundancy will occur, where there is no effective reconcilability. Creating large amounts of redundant detailed data is a very undesirable condition for the detailed level of data in the data warehouse and defeats its purpose. If multiple development groups will be doing concurrent design and population in the current level of detail, great care must be taken to ensure that no redundant detailed data is created.

To ensure that no redundant data is developed, it is necessary to create a data model that reflects the common detailed data. Figure 6.32 shows that multiple development groups have combined their interests to create a common data model. In addition to the currently active development groups, other groups that will have future requirements but who are not currently in a development mode may also contribute their requirements. (Of course, if a group knows it will have future requirements but is unable to articulate them, then those requirements cannot be factored into the common detailed data model.) The common

detailed data model reflects the collective need among the different groups for detailed data in the data warehouse.

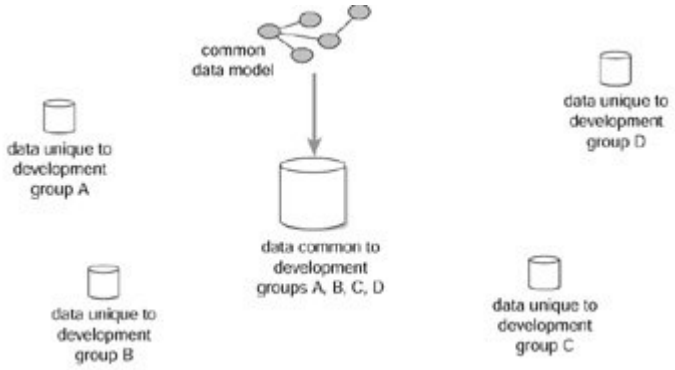

Figure 6.32: A data model identifies data that is common to all the development groups.

The data model forms the basis of the design for the data warehouse. Figure 6.33 shows that the data model will be broken up into many tables as design progresses, each of which physically becomes part of the warehouse.

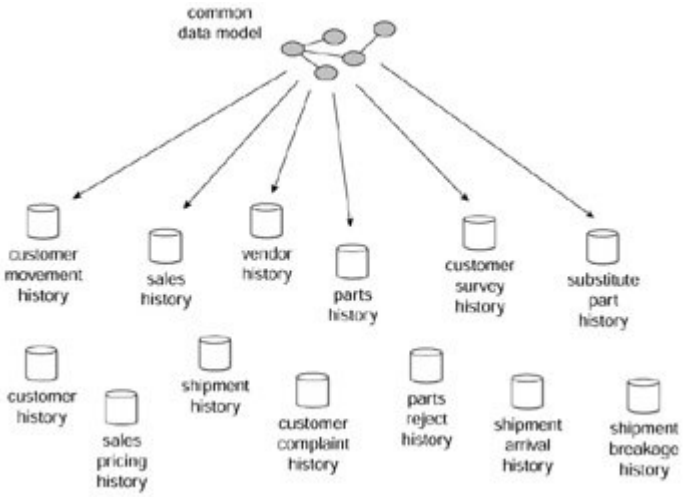

Figure 6.33: The data warehouse is physically manifested over multiple physical tables and databases.

Because the data model is broken into multiple physical tables at the moment of implementation, the development process for the data warehouse can proceed in an iterative manner. There is no need to build all of the tables at once. In fact, a good reason to build only a few tables at a time is so that the end user feedback can be factored into the modification of the table, if necessary, with a minimum of fuss. In addition, because the common data model is broken into multiple tables, adding new tables at a later time to reflect requirements that are now unknown is not a problem.

Different Requirements at Different Levels

Normally different groups have unique requirements (see Figure 6.34). These requirements result in what can be termed “local” current-level detail. The local data is certainly part of the

data warehouse. It is, however, distinctively different from the “common” part. The local data has its own data model, usually much smaller and simpler than the common detailed data model.

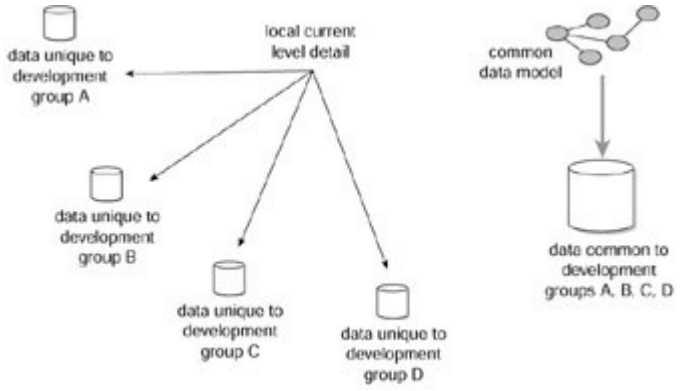

Figure 6.34: Just because data is not common to all development groups does not mean that it does not belong in the current-level detail of the data warehouse.

There is, of necessity, nonredundancy of data across all of the detailed data. Figure 6.35 makes this point clear.

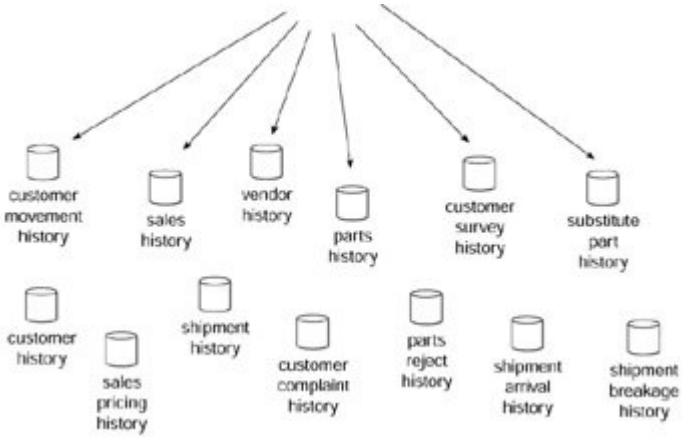

Figure 6.35: Nonredundancy of nonkey data throughout the many tables that make up the detailed level of the data warehouse.

Of course, the nonredundancy of the data is restricted to nonkey data. Redundancy exists at the key level because a form of foreign key relationships is used to relate the different types of data. Figure 6.36 shows the use of foreign keys.

The foreign keys found in the tables shown in Figure 6.36 are quite different from the classical foreign key relationships that are governed by referential integrity. Because the data in the data warehouse is gathered by and stored in terms of snapshots of data, the foreign key relationships that are found are organized in terms of “artifacts” of relationships. For an in-depth explanation of artifacts of relationships, refer to the *www.billinmon.com* Tech Topic on the subject, found in the “References” section.

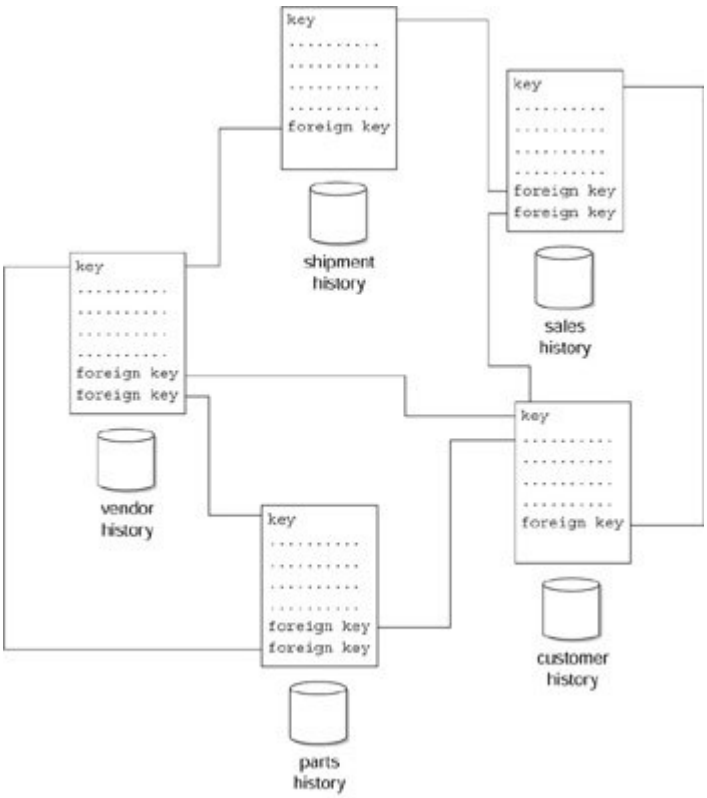

Figure 6.36: Foreign keys in the data warehouse environment.

An issue that arises is whether to place all of the detailed tables—common and local—under the same technology, as shown in Figure 6.37. There are many good arguments for doing so One is that the cost of a single platform versus multiple platforms is much less. Another is that the cost of support and training will be less. In fact, about the only argument for multiple platforms for detailed data is that with multiple platforms, there may not be the need for a single massively large platform, and as a consequence, the cost of the multiple smaller platforms may be less than a single larger platform. In any case, many organizations adopt the strategy of a single platform for all their detailed data warehouse data, and the strategy works well.

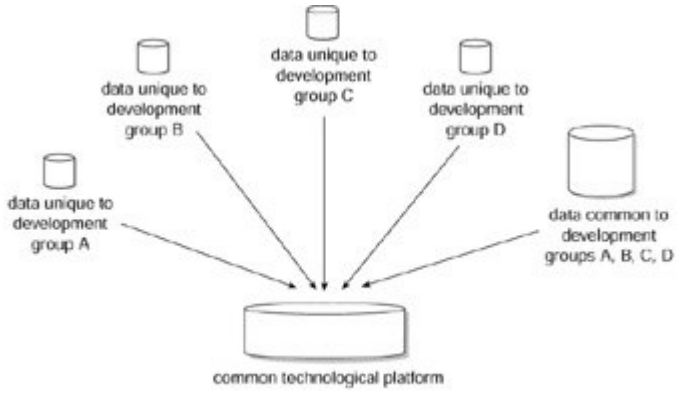

Figure 6.37: The different types of data in the detailed level of the data warehouse all on a common platform.

Other Types of Detailed Data

Another strategy is to use different platforms for the different types of data found at the detailed level. Figure 6.38 shows one example of this option. Some of the local data is on one platform, the common data is on another platform, and other local data is on yet another. This option is certainly one that is valid, and it often satisfies the different political needs of the organization. With this option each group doing development can feel that it has some degree of control of at least its own peculiar needs. Unfortunately, this option has several major drawbacks. First, multiple technologies must be purchased and supported. Second, the end user needs to be trained in different technologies. And finally, the boundaries between the technologies may not be as easy to cross. Figure 6.39 illustrates this dilemma.

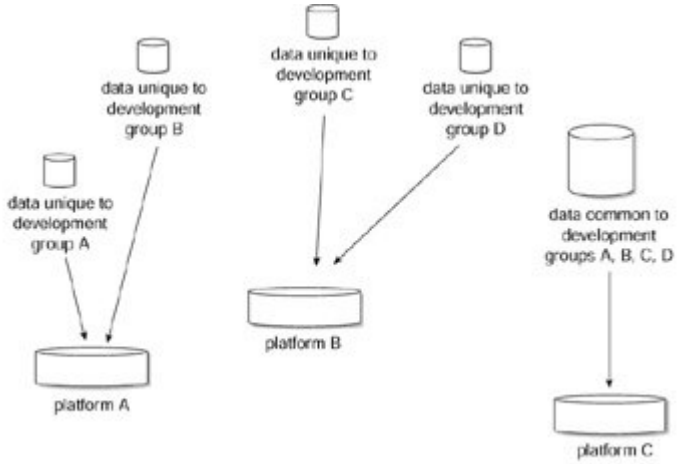

Figure 6.38: In this case, the different parts of the detailed level of the data warehouse are scattered across different technological platforms.

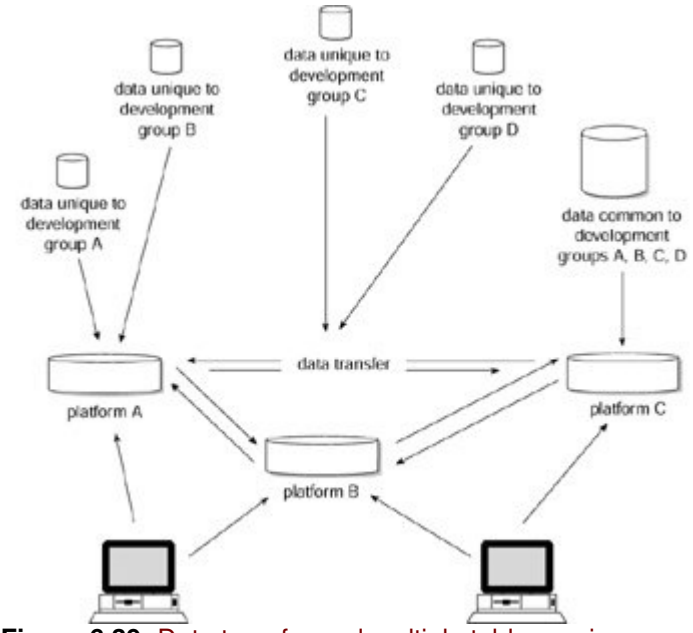

Figure 6.39: Data transfer and multiple table queries present special technological problems.

If there are to be multiple technologies supporting the different levels of detail in the data warehouse, it will be necessary to cross the boundaries between the technologies frequently. Software that is designed to access data across different technological platforms is available. Some of the problems that remain are shown in Figure 6.40.

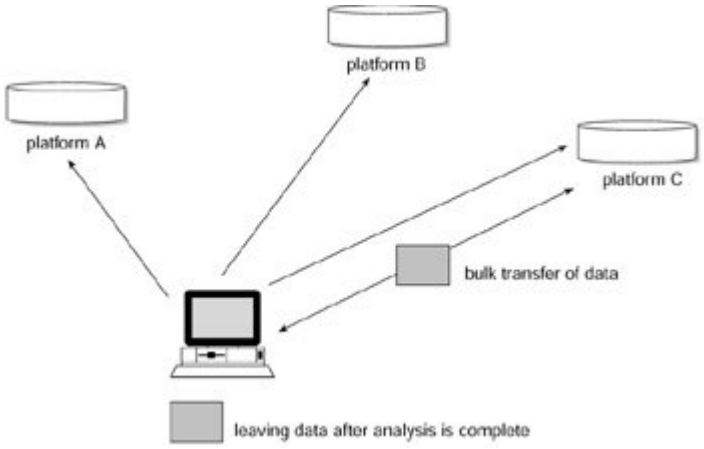

Figure 6.40: Some problems with interfacing different platforms.

One problem is in the passage of data. If multi-interfaced technology is used for the passage of small amounts of data, then there is no problem with performance. But if multi-interfaced technology is used to pass large amounts of data, then the software can become a performance bottleneck. Unfortunately, in a DSS environment it is almost impossible to know how much data will be accessed by any one request. Some requests access very little data; other requests access large amounts of data. This problem of resource utilization and management manifests itself when detailed data resides on multiple platforms.

Another related problem is “leaving” detailed data on one side of the data warehouse after it has been transported from the other side. This casual redeployment of detailed data has the effect of creating redundancy of data at the detailed level, something that is not acceptable.

Meta Data

In any case, whether detailed data is managed on a single technology or on multiple technologies, the role of meta data is not diminished. Figure 6.41 shows that meta data is needed to sit on top of the detailed data warehouse data.

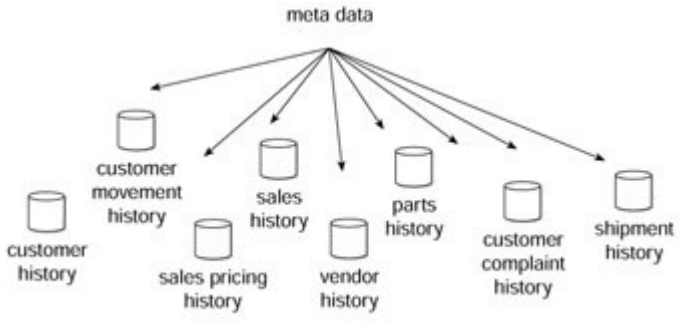

Figure 6.41: Meta data sits on top of the actual data contents of the data warehouse.

Multiple Platforms for Common Detail Data

One other possibility worth mentioning is using multiple platforms for common detail of data. Figure 6.42 outlines this scenario.

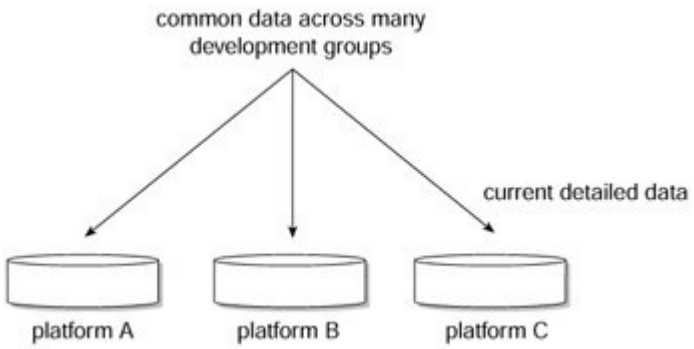

Figure 6.42: Common detailed data across multiple platforms-a real red flag in all cases.

While such a possibility is certainly an option, however, it is almost never a good choice. Managing common current detailed data is difficult enough. The volumes of data found at that level present their own unique problems for management. Adding the complication of having to cross multiple technological platforms merely makes life more difficult. Unless there are very unusual mitigating circumstances, this option is not recommended.

The only advantage of multiple platforms for the management of common detail is that this option satisfies immediate political and organizational differences of opinion.

Summary

Most environments operate from a single centralized data warehouse. But in some circumstances there can be a distributed data warehouse. The three types of distributed data warehouses are as follows:

- Data warehouses serving global businesses where there are local operations and a central operation
- Technologically distributed data warehouses where the volume of data is such that the data is spread over multiple physical volumes
- Disparate data warehouses that have grown separately through lack of organizational or political alignment

Each type of distributed data warehouses has its own considerations.

The most difficult aspect of a global data warehouse is the mapping done at the local level. The mapping must account for conversion, integration, and different business practices. The mapping is done iteratively. In many cases, the global data warehouse will be quite simple because only the corporate data that participates in business integration will be found in the global data warehouse. Much of the local data will never be passed to or participate in the loading of the global data warehouse. Access of global data is done according to the business needs of the analyst. As long as the analyst is focusing on a local business practice, access to global data is an acceptable practice.

The local data warehouses often are housed on different technologies. In addition, the global data warehouse may be on a different technology than any of the local data warehouses. The corporate data model acts as the glue that holds the different local data warehouses

together, as far as their intersection at the global data warehouse is concerned. There may be local data warehouses that house data unique to and of interest to the local operating site. There may also be a globally distributed data warehouse. The structure and content of the distributed global data warehouse are determined centrally, whereas the mapping of data into the global data warehouse is determined locally.

The coordination and administration of the distributed data warehouse environment is much more complex than that of the single-site data warehouse. Many issues relate to the transport of the data from the local environment to the global environment, including the following questions:

- What network technology will be used?
- Is the transport of data legal?
- Is there a processing window large enough at the global site?
- What technological conversion must be done

Chapter 7: Executive Information Systems and the Data Warehouse

Overview

Prior to data warehousing, there were executive information systems (EIS). EIS was a notion that computation should be available to everyone, not just the clerical community doing day-to-day transactions. EIS presented the executive with a set of appealing screens. The idea was that the elegance of the screen presentation would beguile the executive. While there certainly is merit to the idea that the world of computation should be open to the executive, the founders of EIS had no concept of the infrastructure needed to get those numbers to the executive. The entire idea behind EIS was presentation of information with no real understanding of the infrastructure needed to create that information in the first place. When the data warehouse first appeared, the EIS community roundly derided it as a complex discipline that required getting the hands dirty. EIS was a high-minded, elegant discipline that was above the hard work and management of complexity involved in a data warehouse. The EIS community decided that executives had better things to do than worry about such issues as sources of data, quality of data, currency of data, and so forth. And so EIS died for lack of an infrastructure. It hardly mattered that the presentation to the executive was elegant if the numbers being presented were unbelievable, inaccurate, or just plain unavailable.

This chapter first appeared just as EIS was on its way out. As originally written, this chapter was an attempt to appeal to the EIS community, based on the rationality of the necessity of an infrastructure. But the wisdom of the EIS community and its venture capital backers was such that there was to be no relationship between data warehousing and EIS. When it came to the infrastructure needed to support the grandiose plans of the EIS community, the EIS community and the venture capital community just didn't get it.

EIS as it was known in its earliest manifestation has all but disappeared. But the promises made by EIS are still valuable and real. Consequently EIS has reappeared in many forms today—such as OLAP processing and DSS applications such as customer relationship management (CRM)—and those more modern forms of EIS are very much related to data warehousing, unlike the earliest forms of EIS.

EIS—The Promise

EIS is one of the most potent forms of computing. Through EIS, the executive analyst can pinpoint problems and detect trends that are of vital importance to management. In a sense, EIS represents one of the most sophisticated applications of computer technology.

EIS processing is designed to help the executive make decisions. In many regards, EIS becomes the executive's window into the corporation. EIS processing looks across broad vistas and picks out the aspects that are relevant to the running of the business. Some of the typical uses of EIS are these:

- Trend analysis and detection
- Key ratio indicator measurement and tracking
- Drill-down analysis
- Problem monitoring
- Competitive analysis
- Key performance indicator monitoring

A Simple Example

As an example of how EIS analysis might appear to an executive, consider Figure 7.1, which shows information on policies offered by an insurance company. Quarter by quarter, the new life, health, and casualty policy sales are tracked. The simple graph shown in Figure 7.1 is a good starting point for an executive's probing into the state of the business. Once the executive has seen the overall information, he or she can probe more deeply, as shown by the trend analysis in Figure 7.2.

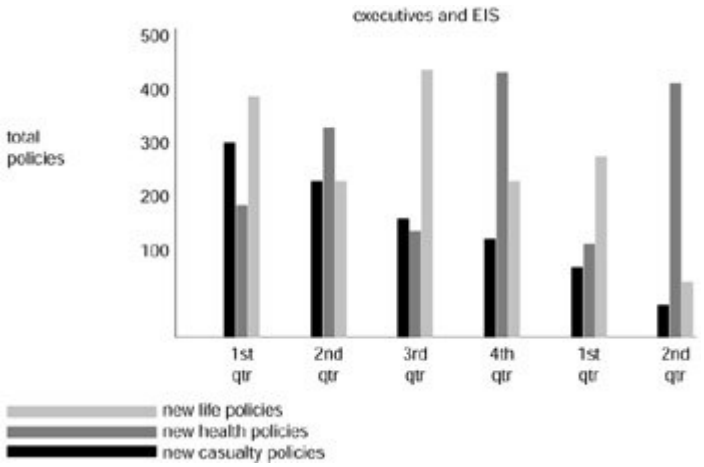

Figure 7.1: A chart typical of EIS processing.

In Figure 7.2, the executive has isolated new casualty sales from new life sales and new health sales. Looking just at new casualty sales, the executive identifies a trend: New casualty sales are dropping off each quarter. Having identified the trend, the executive can investigate why sales are dropping.

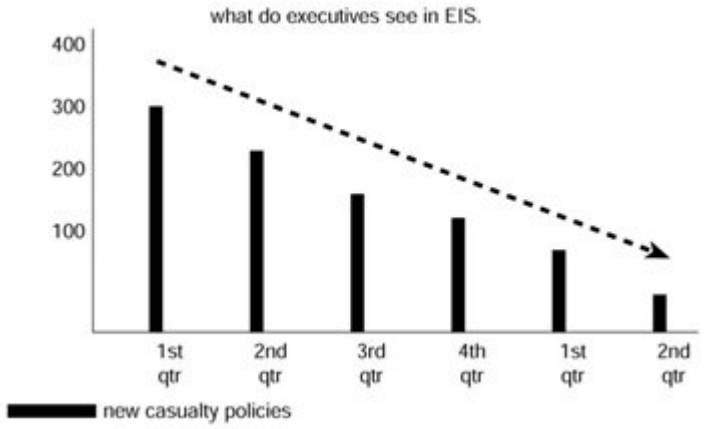

Figure 7.2: Trends—new casualty policy sales are dropping off.

The EIS analysis alerts the executive as to what the trends are. It is then up to him or her to discover the underlying reasons for the trends.

The executive is interested in both negative and positive trends. If business is getting worse, why, and at what rate? What can be done to remedy the situation? Or, if business is picking

up, who and what are responsible for the upturn? What can be done to accelerate and accentuate the success factors? Can the success factors be applied to other parts of the business?

Trends are not the only type of analysis accommodated by EIS. Another type of useful analysis is comparisons. Figure 7.3 shows a comparison that might be found in an EIS analysis.

Looking at fourth-quarter data, first-quarter data, and second-quarter data in Figure 7.3, the question can be asked, "Why is there such a difference in sales of new health policies for the past three quarters?" The EIS processing alerts the manager to these differences. It is then the job of the EIS analyst to determine the underlying reasons.

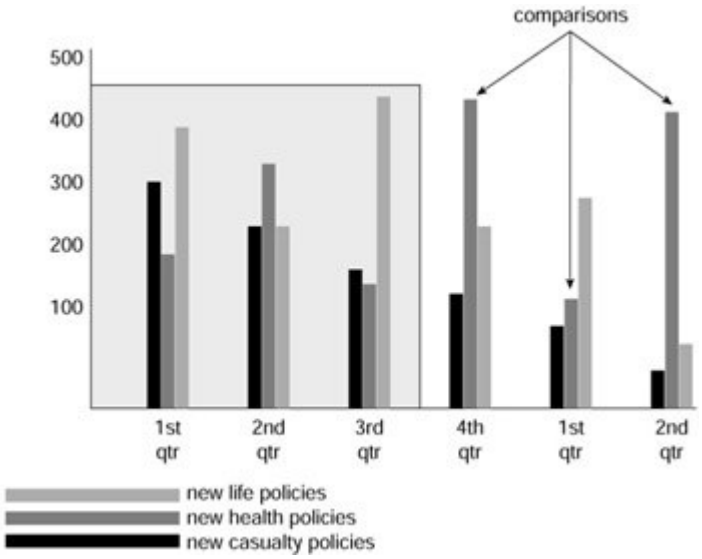

Figure 7.3: Why is there an extreme difference in sales of new health policies for the past three quarters?

For the manager of a large, diverse enterprise, EIS allows a look at the activities of the enterprise in many ways. Trying to keep track of a large number of activities is much more difficult than trying to keep track of just a few activities. In that sense, EIS can be used to expand the scope of control of a manager.

But trend analysis and comparisons are not the only ways that the manager can use EIS effectively. Another approach is to "slice-and-dice." Here the analyst takes basic information, groups it one way, and analyzes it, then groups it another way and reanalyzes it. Slicing and dicing allows the manager to have many different perspectives of the activities that are occurring.

Drill-Down Analysis

To do slicing and dicing, it is necessary to be able to "drill down" on data. Drilling down refers to the ability to start at a summary number and to break that summary into a successively finer set of summarizations. By being able to get at the detail beneath a summary number, the manager can get a feel for what is happening, especially where the summary number is surprising. Figure 7.4 shows a simple example of drill-down analysis.

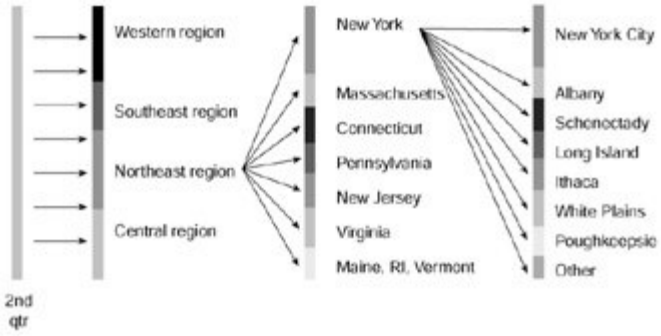

Figure 7.4: To make sense of the numbers shown by EIS, the numbers need to support a drill-down process.

In Figure 7.4, the manager has seen second-quarter summary results and wants to explore them further. The manager then looks at the regions that have contributed to the summary analysis. The figures analyzed are those of the Western region, the Southeast region, the Northeast region, and the Central region. In looking at the numbers of each region, the manager decides to look more closely at the Northeast region's numbers.

The Northeast's numbers are made up of totals from New York, Massachusetts, Connecticut, Pennsylvania, New Jersey, Virginia, Maine, Rhode Island, and Vermont. Of these states, the manager then decides to look more closely at the numbers for New York state. The different cities in New York state that have outlets are then queried.

In each case, the manager has selected a path of going from summary to detail, then to a successively lower level. In such a fashion, he or she can determine where the troublesome results are. Once having identified the anomalies, the manager then knows where to look more closely.

Yet another important aspect of EIS is the ability to track key performance indicators. Although each corporation has its own set, typical key performance indicators might be the following:

- Cash on hand
- Customer pipeline
- Length of sales cycle
- Collection time
- New product channel
- Competitive products

Each corporation has several key performance indicators that—in a single measurement—tell an important story about some aspect of the life of the corporation. On their own, the key performance indicators say a lot about what is going on in the corporation. Taken over time, the key performance indicators say even more because they indicate trends.

It is one thing to say that cash on hand is $X. It is even more powerful to say that two months ago cash on hand was $Z, one month ago cash on hand was $Y, and this month cash on hand is $X. Looking at key performance indicators over time is one of the most important things an executive can do, and EIS is ideal for this purpose.

There is plenty of very sophisticated software that can be used in EIS to present the results to a manager. The difficult part of EIS is not in the graphical presentation but in discovering and preparing the numbers that go into the graphics, as seen in Figure 7.5.

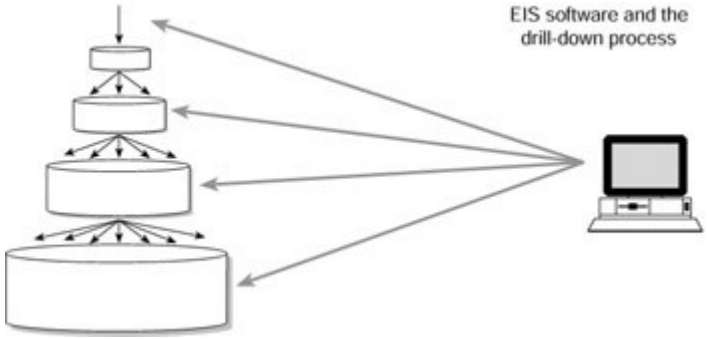

Figure 7.5: EIS software supports the drill-down process as long as the data that is needed is available and is structured properly.

EIS is perfectly capable of supporting the drill-down process from the graphical perspective as long as the data exists in the first place. However, if the data to analyze does not exist, the drill-down process becomes very tedious and awkward, certainly not something the executive wants to do.

Supporting the Drill-Down Process

Creating the basis of data on which to perform drill-down analysis, then, is the major obstacle to successfully implementing the drill-down process, as seen in Figure 7.6. Indeed, some studies indicate that $9 is spent on drill-down data preparation for every $1 spent on EIS software and hardware.

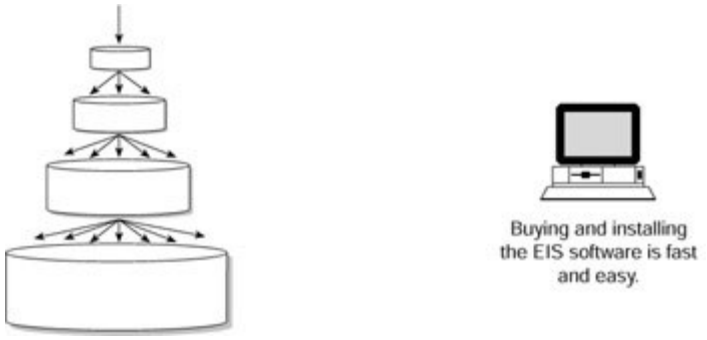

Figure 7.6: Creating the base of data on which to do EIS is the hard part.

Exacerbating the problem is the fact that the executive is constantly changing his or her mind about what is of interest, as shown in Figure 7.7. On day 1, the executive is interested in the corporation's financial activities. The EIS analyst makes a big effort to develop the underlying data to support EIS interest. Then on day 2, there is an unexpected production problem, and management's attention turns there. The EIS analyst scurries around and tries to gather the data needed by the executive. On day 3, the EIS analyst is directed to the problems that have developed in shipping. Each day there is a new focus for the executive. The EIS analyst simply cannot keep up with the rate at which the executive changes his or her mind.

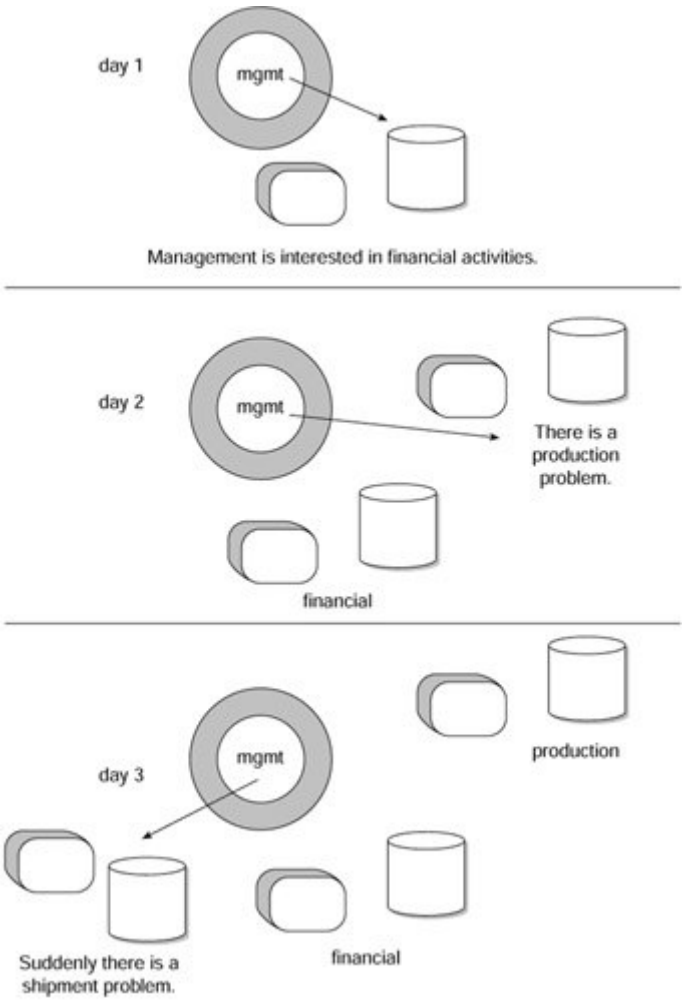

Figure 7.7: The constantly changing interests of executives.

Management's focus in the running of the business shifts with every new problem or opportunity that arises. There simply is no predictable pattern for what management will be interested in tomorrow. In turn, the EIS analyst is at the end of a whip—the wrong end! The EIS analyst is forever in a reactive state. Furthermore, given the work that is required of the EIS analyst to build the base of data needed for EIS analysis, the EIS analyst is constantly swamped.

The problem is that there is no basis of data from which the EIS analyst can easily work. Each new focus of management requires an entirely different set of data for the EIS analyst. There is no infrastructure to support the EIS environment.

The Data Warehouse as a Basis for EIS

It is in the EIS environment that the data warehouse operates in its most effective state. The data warehouse is tailor-made for the needs of the EIS analyst. Once the data warehouse has been built, the job of the EIS is infinitely easier than when there is no foundation of data on which the EIS analyst can operate. Figure 7.8 shows how the data warehouse supports the need for EIS data.

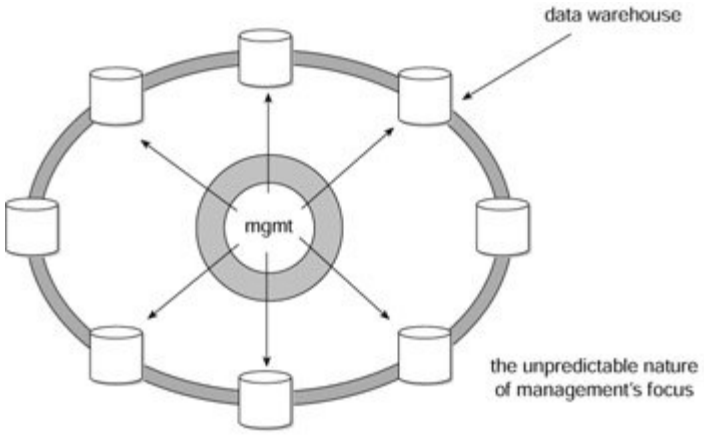

Figure 7.8: The data warehouse supports management's need for EIS data.

With a data warehouse, the EIS analyst does not have to worry about the following:

- Searching for the definitive source of data
- Creating special extract programs from existing systems
- Dealing with unintegrated data
- Compiling and linking detailed and summary data and the linkage between the two
- Finding an appropriate time basis of data (i.e., does not have to worry about finding historical data)
- Management constantly changing its mind about what needs to be looked at next

In addition, the EIS analyst has a rich supply of summary data available.

In short, the data warehouse provides the basis of data-the infrastructure—that the EIS analyst needs to support EIS processing effectively. With a fully populated data warehouse in place, the EIS analyst can be in a proactive stance—not an eternally reactive stance—with regard to answering management's needs. The EIS analyst's job changes from that of playing data engineer to that of doing true analysis, thanks to the data warehouse.

Yet another very important reason why the data warehouse serves the needs of the world of EIS is this: The data warehouse operates at a low level of granularity. The data warehouse contains—for lack of a better word—atomic data. The atomic data can be shaped one way, then another. When management has a new set of needs for information that has never before been encountered in the corporation, the very detailed data found in the data warehouse sits, waiting to be shaped in a manner suited to management's needs. Because of the granular atomic data that resides in the data warehouse, analysis is flexible and responsive. The detailed data in the data warehouse sits and waits for future unknown needs for information. This is why the data warehouse turns an organization from a reactive stance to a proactive stance.

Where to Turn

The EIS analyst can turn to various places in the architecture to get data. In Figure 7.9, the EIS analyst can go to the individual level of processing, the departmental (data mart) level of processing, the lightly summarized level of processing, or the archival/dormant level of data. In addition, there is a normal sequence or hierarchy in which the EIS analyst goes after data to serve management's needs (see Figure 7.9).

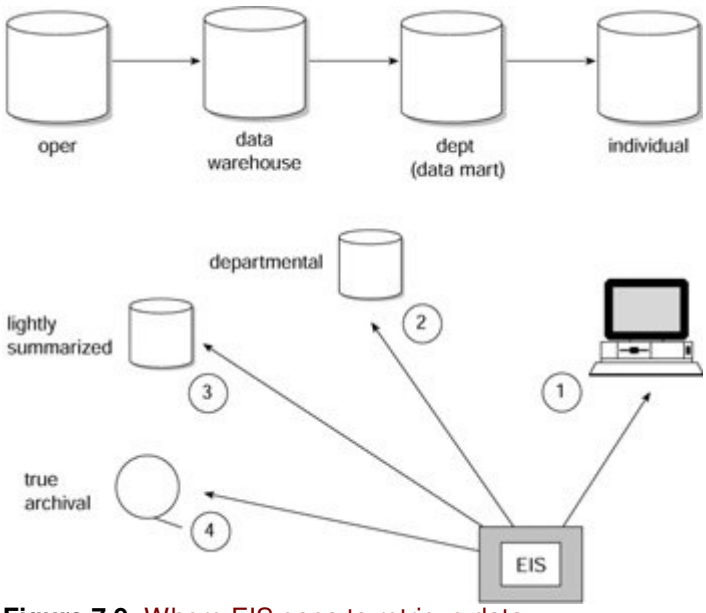

Figure 7.9: Where EIS goes to retrieve data.

There is a very good reason for the order shown, as indicated in Figure 7.10. By going from the individual level of processing to the archival/dormant level, the analyst does de facto drill-down analysis. The most summarized data found in the architected environment is at the individual level. The supporting level of summary for the individual level is the departmental (data mart) level. Supporting the summaries at the departmental (data mart) level is data at the data warehouse lightly summarized level. Finally, the light summarization at the data warehouse level is supported by archival/dormant data. The sequence of summaries just described is precisely what is required to support drill-down EIS analysis.

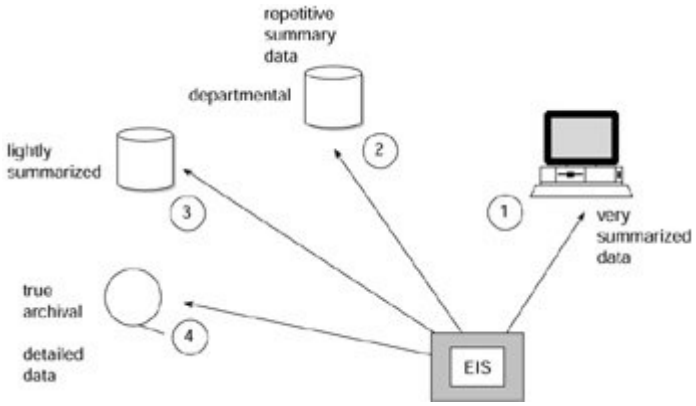

Figure 7.10: In going from individual levels of processing to true archival data, the drill-down process is accommodated.

Almost by default, the data warehouse lays a path for drill-down analysis. At the different levels of the data warehouse, and throughout the summarization process, data is related by means of a key structure. The key structure itself, or the derivations of the key structure, allow the data to be related so that drill-down analysis is easy and natural.

The ways that EIS is supported by the data warehouse are illustrated in Figure 7.11.

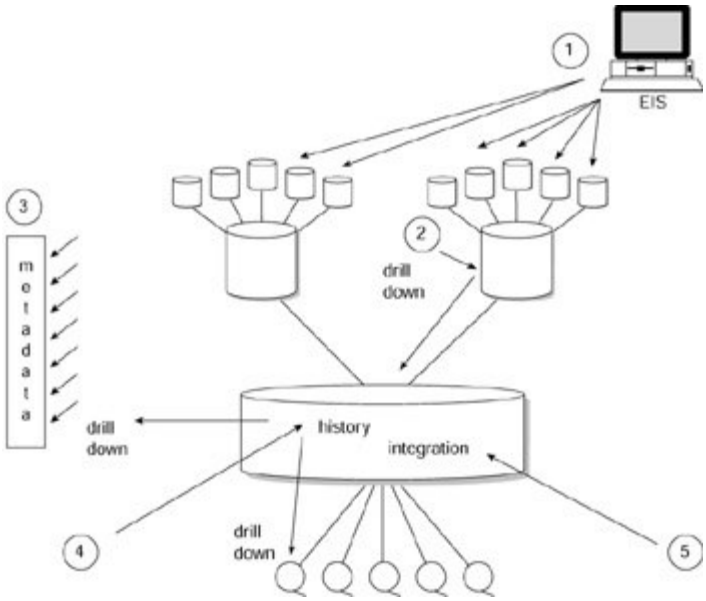

Figure 7.11: How EIS is supported by the data warehouse.

The EIS function uses the following:

- The data warehouse for a readily available supply of summary data
- The structure of the data warehouse to support the drill-down process
- Data warehouse meta data for the DSS analyst to plan how the EIS system is built
- The historical content of the data warehouse to support the trend analysis that management wishes to see
- The integrated data found throughout the data warehouse to look at data across the corporation

Event Mapping

A useful technique in using the data warehouse for EIS processing is event mapping. The simplest way to depict event mapping is to start with a simple trend line.

Figure 7.12 shows that corporate revenues have varied by month, as expected. The trend has been calculated from data found in the data warehouse. The trend of revenues in and of itself is interesting but gives only a superficial view of what is going on with the corporation. To enhance the view, events are mapped onto the trend line.

Figure 7.12: Simple trends.

In Figure 7.13, three notable events have been mapped to the corporate revenue trend line—the introduction of a “spring colors” line of products, the advent of a sales incentive program, and the introduction of competition. Now the relationship between corporate

revenue and significant events begins to take on a different perspective. Looking at the diagram in Figure 7.13, one might draw the conclusion that the introduction of a new line of products and a new incentive plan have boosted revenue and that competition is starting to have an effect in the latter part of the year.

Figure 7.13: Mapping events against a trend line.

For some sorts of events, event mapping is the only way to measure the results. Some events and activities cannot be measured directly and have to be measured in a correlative fashion. Cost justification and actual cost benefit cannot be measured any other way for some types of events.

Misleading conclusions can be drawn, though, by looking at correlative information. It often helps to look at more than one set of trends that relate to the events at hand. As an example, Figure 7.14 shows that corporate revenues are matched against the consumer confidence index to produce a diagram packed with even more perspective. Looking at the figure shown, the executive can make up his or her own mind whether events that have been mapped have shaped sales.

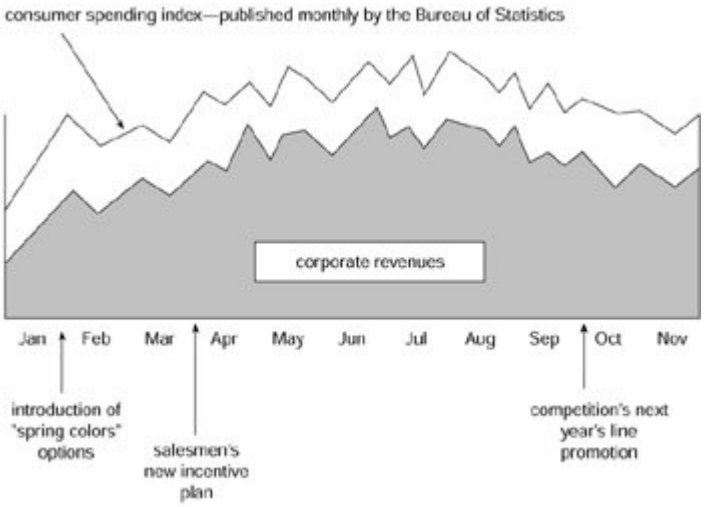

Figure 7.14: Superimposing another trend analysis over the existing one to gain another perspective.

The data warehouse can store both the internally generated revenue numbers and the externally generated consumer confidence

Detailed Data and EIS

Just how much detailed data do you need to run your EIS/DSS environment? One school of thought says that you need as much detail as possible. By storing as much data as possible, you can do any kind of analysis that might happen along. Because the nature of DSS is delving into the unknown, who knows what detail you will need? To be on the safe side, you'd better keep all the detailed data you can get your hands on. Furthermore, the more historical detailed data you can get, the better, because you never can tell how far back you need to go to do a given DSS analysis.

The logic at the heart of the argument for the storage of massive amounts of detail for DSS processing is hard to argue with. Intellectually, it must be correct to say that you need as much detail as possible for DSS and EIS processing. But, in some important ways, the argument suggests Zeno's paradox. In Zeno's paradox, logic inescapably "proves" that a rabbit can never outrun a turtle as long as the turtle has a head start on the rabbit. Of course, reality and our own observations tell us something quite different, warning us that any conclusion based purely on logic is circumspect.

What, then, is so wrong with keeping all the detail in the world around when you are building a DSS/EIS environment? There are several things wrong. First, the amount of money required for both storage and processing costs can go sky-high. The sheer cost of storing and processing huge amounts of detailed data prohibits the establishment of an effective EIS/DSS environment. Second, massive amounts of data form an obstacle to the effective use of analysis techniques. Given very large amounts of data to be processed, important trends and patterns can hide behind the mask of endless records of detailed data. Third, with the detail, reuse of previous analysis is not fostered. As long as there is a massive amount of detail around, DSS analysts are encouraged to create new analyses from scratch. Such a practice is wasteful and potentially harmful. When new analysis is not done in quite the same way as older analysis, very similar analyses are done and ironically conflicting results are obtained.

There is, then, a very real case for storing summary data as well as detailed data. DSS and EIS processing ought to make as much use of summary data as they do of detailed data. Summary data is much less voluminous and much easier to manage than detailed data. From an access and presentation perspective, summary data is ideal for management. Summary data represents a foundation on which future analysis can build and for which existing analysis does not have to be repeated. For these reasons alone, summary data is an integral part of the DSS/EIS environment.

Keeping Only Summary Data in the EIS

Some very real problems become evident with keeping just summary data. First, summary data implies a process—the summary is always created as a result of the process of calculation. The calculation may be very simple or complex. In any case, there is no such thing as summary data that stands alone—summary data of necessity stands with its process. To effectively use summary data, the DSS analyst must have access to and an understanding of the process that has been used to shape it. As long as DSS and EIS understand this relationship between process and summary data and as long as EIS and DSS can profitably use the summary data that has resulted from the process of calculation, then summary data constitutes an ideal foundation for EIS and DSS. However, if the analysts that are doing EIS/DSS analysis do not understand that process is intimately related to summary data, the results of the analysis can be very misleading.

The second problem with summary data is that it may or may not be at the appropriate level of granularity for the analytical purpose at hand. A balance needs to be struck between the level of detail and the level of summarization for EIS and DSS processing.

Summary

There is a very strong affinity between the needs of the EIS analyst and the data warehouse. The data warehouse explicitly supports all of the EIS analyst's needs. With a data warehouse in place, the EIS analyst can be in a proactive rather than a reactive position.

The data warehouse enables the EIS analyst to deal with the following management needs:

- Accessing information quickly
- Changing their minds (i.e., flexibility)
- Looking at integrated data
- Analyzing data over a spectrum of time
- Drilling down

The data warehouse provides an infrastructure on which the EIS analyst can build.

Chapter 8: External/Unstructured Data and the Data Warehouse

Overview

Most organizations build their first data warehouse efforts on data whose source is existing systems (i.e., on data internal to the corporation). In almost every case, this data can be termed internal, structured data. The data comes internally from the corporation and has been already shaped into a regularly occurring format.

A whole host of other data is of legitimate use to a corporation that is not generated from the corporation's own systems. This class of data is called external data and usually enters the corporation in an unstructured, unpredictable format. Figure 8.1 shows external and unstructured data entering the data warehouse.

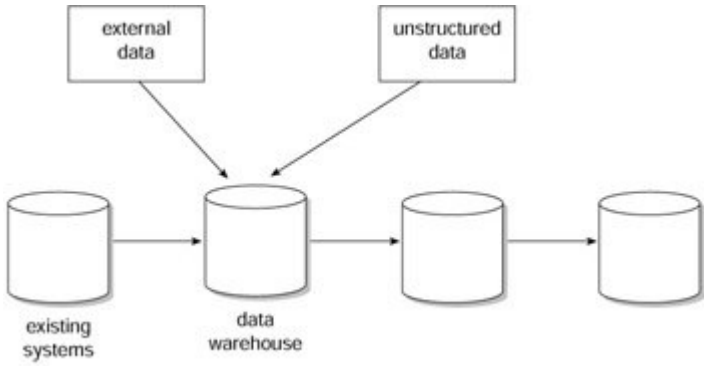

Figure 8.1: External and unstructured data both belong in the data warehouse.

The data warehouse is the ideal place to store external and unstructured data. If external and unstructured data is not stored in a centrally located place, several problems are sure to arise. Figure 8.2 shows that when this type of data enters the corporation in an undisciplined fashion, the identity of the source of the data is lost, and there is no coordination whatsoever in the orderly use of the data.

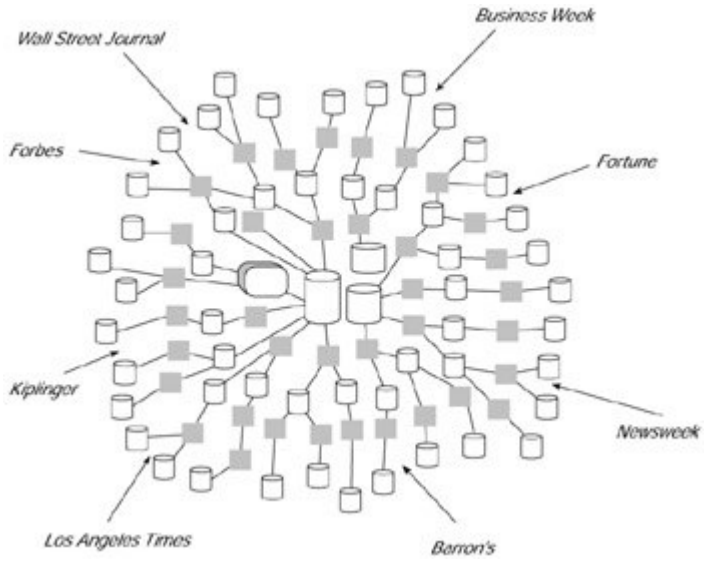

Figure 8.2: Problems with unstructured data.

Typically, when external data is not entered into the data warehouse, it comes into the corporation by means of the PC. There is nothing wrong per se with entering data at the PC level. But almost always, the data is entered manually through a spreadsheet, and absolutely no attempt to is made capture information about its source. For example, in Figure 8.2 an analyst sees a report in the *Wall Street Journal*. The next day, the analyst uses the data from the *Wall Street Journal* as part of a report, but the original source of the data is lost as it is entered into the corporate mainstream of data.

Another difficulty with the laissez-faire approach to external data is that at a later time it is hard to recall the data. It is entered into the corporation's systems, used once, and then it disappears. Even a few weeks later, it is hard to find and then reprocess the data for further use. This is unfortunate because much of the data coming from external sources is quite useful over the spectrum of time.

The types of data from external sources are many and diverse. Some typical sources of interesting data include the following:

- *Wall Street Journal*
- *Business Week*
- Forbes
- Fortune
- Industry newsletters
- Technology reports
- Dun & Bradstreet (now D&B)
- Reports generated by consultants specifically for the corporation
- Equifax reports
- Competitive analysis reports
- Marketing comparison and analysis reports
- Sales analysis and comparison reports
- New product announcements

In addition, reports internal to the corporation are of interest as well:

- Auditor's quarterly report
- Annual report
- Consultant reports

In a sense, the data generated by the Web-based ebusiness environment is unstructured. It is at such a low level of detail that the data must be reconstituted before it is useful. This clickstream data then is merely a sophisticated form of unstructured data.

External/Unstructured Data in the Data Warehouse

Several issues relate to the use and storage of external and unstructured data in the data warehouse. One problem of unstructured data is the frequency of availability. Unlike internally appearing data, there is no real fixed pattern of appearance for external data. This irregularity is a problem because constant monitoring must be set up to ensure that the right data is captured. For some environments, such as the Internet, monitoring programs can be created and used to build automated alerts.

The second problem with external data is that it is totally undisciplined. To be useful, and for placement in the warehouse, a certain amount of reformatting of external data is needed to transform it into an internally acceptable and usable form. A common practice is to convert the external data as it enters the data warehouse environment. External key data is converted to internal key data. Or external data is passed through simple edits, such as a domain check. In addition, the data is often restructured so that it is compatible with internal data.

In some cases, the level of granularity of the external data will not match that of the internal systems of the corporation. For example, suppose a corporation has individual household information. Now suppose the corporation purchases a list of income by zip code. The external list says that the average household income in the zip code is $X. The matching of the internal household information is done such that each household in a zip code is assigned the income specified by the external file. (This means that some households will be assigned an income level below their means and that other households will be assigned an income level above their means. But, on average, the household income will be about right.) Once this arbitrary assignment of income is done, the data can be sliced and diced into many other patterns.

The third factor that makes external data hard to capture is its unpredictability. External data may come from practically any source at almost any time.

In addition to external data that might come from a magazine article or a consultant's report, another whole class of data is just now able to be automated—unstructured data. The two most common types of unstructured data are image and voice data. Image data is stored as pictures; voice data is stored digitally and can be translated back into a voice format. The issues of image data and voice data stem primarily from technology. The technology to capture and manipulate image and voice data is not nearly as mature as more conventional technology. In addition, even when image and voice data can be captured, their storage requires huge amounts of DASD, and their recall and display or playback can be awkward and slow.

Nevertheless, there are many methods to capture and store unstructured information. One of the best ways is to place it on some bulk storage medium such as near-line storage. With near-line storage the unstructured data is still accessible but it doesn't cost huge amounts of money to store. Of course, extensive indexing of the unstructured data can be done, and those indexes can be stored in both disk storage and near-line storage. In such a manner many requests about the unstructured data can be managed without actually having to go to the unstructured data. In addition, some requests can be handled entirely inside the index of the unstructured data itself. Also, if an extensive index of unstructured data is created, the unstructured data can be tied to structured data. The index can then be used to determine what unstructured data to bring to disk storage. In this case, only unstructured data that is prequalified and preselected would be brought to disk storage.

Another technique for handling unstructured data that is sometimes effective is to create two stores of unstructured data. One store contains all of the unstructured data, and another, much smaller store contains only a subset. The subset of unstructured data can be accessed and analyzed before the large, complete unstructured store is analyzed. When this is done, the potential to save on processing is huge.

The unstructured data becomes an adjunct to the data warehouse. The unstructured data is connected to the data warehouse by means of an index, and the unstructured data is brought into the data warehouse only when there is a specific, prequalified request for it.

Meta Data and External Data

As we've discussed, meta data is an important component of the data warehouse in any scenario, but it takes on an entirely different dimension in the face of storing and managing external and unstructured data. Figure 8.3 shows the role of meta data.

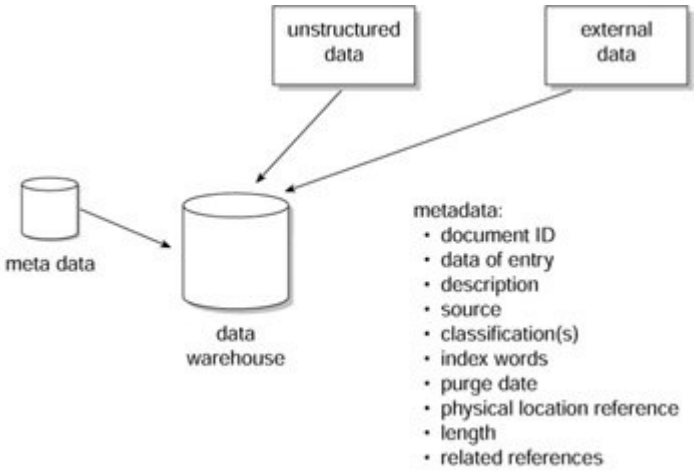

Figure 8.3: Meta data takes on a new role in the face of external and unstructured data.

Meta data is vital because through it external data is registered, accessed, and controlled in the data warehouse environment. The importance of meta data is best understood by noting what it typically encompasses:

- Document ID
- Date of entry into the warehouse
- Description of the document
- Source of the document
- Date of source of the document
- Classification of the document
- Index words
- Purge date
- Physical location reference
- Length of the document
- Related references

It is through the meta data that a manager determines much information about the external data. In many instances, the manager will look at the meta data without ever looking at the source document. Scanning meta data eliminates much work because it filters out documents that are not relevant or are out-of-date. Therefore, properly built and maintained meta data is absolutely essential to the operation of the data warehouse-particularly with regard to external data.

In association with meta data is another type of data-notification data. Shown in Figure 8.4, notification data is merely a file created for users of the system that indicates classifications of data interesting to the users. When data is entered into the data warehouse and into the meta data, a check is made to see who is interested in it. The person is then notified that the external data has been captured.

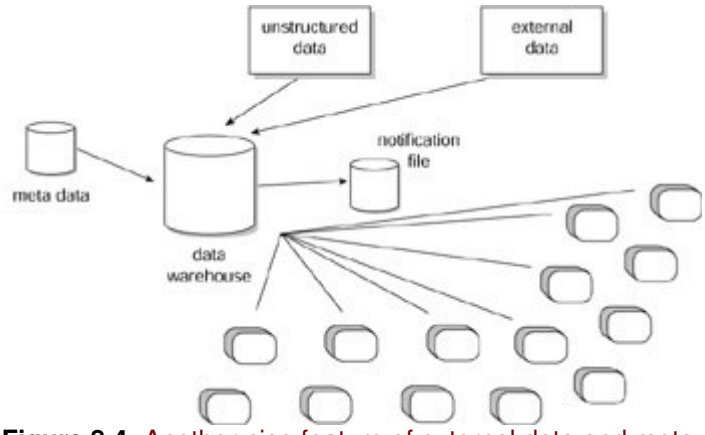

Figure 8.4: Another nice feature of external data and meta data is the ability to create a tailored notification file.

Storing External/Unstructured Data

External data and unstructured data can actually be stored in the data warehouse if it is convenient and cost-effective to do so. But in many cases it will not be possible or economical to store all external data in the data warehouse. Instead, an entry is made in the meta data of the warehouse describing where the actual body of external data can be found. The external data is then stored elsewhere, where it is convenient, as shown in Figure 8.5. External data may be stored in a filing cabinet, on fiche, on magnetic tape, and so on.

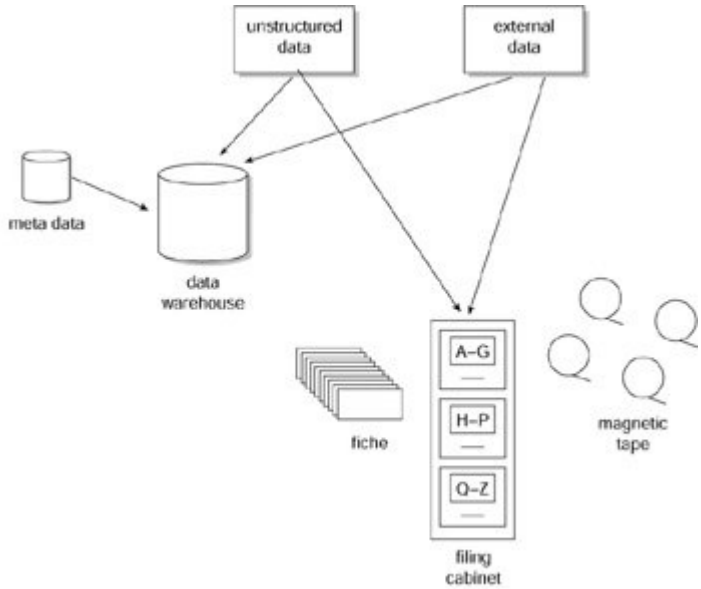

Figure 8.5: In every case, external/unstructured data is registered with meta data, but the actual data may or may not be stored in the data warehouse based on the size of the data and the probability of access.

However it is done, storing external data and unstructured data requires considerable resources. By associating external data and unstructured data with a data warehouse, the external data and the unstructured data become available for all parts of the organization, such as finance, marketing, accounting, sales, engineering, and so forth. The implication is that once the data is captured and managed centrally, the organization has to undergo the expense of dealing with such data only once. But when external data and unstructured data is not associated with a data warehouse, then there is the very real chance that different parts of the organization will capture and store the same data. This duplication of effort and resources is very wasteful and carries with it a very high price tag.

Different Components of External/Unstructured Data

One of the important design considerations of external/unstructured data is that it often contains many different components, some of which are of more use than others. As an example, suppose the complete manufacturing history of a product is purchased. Certain aspects of production are very important, such as the length of time from first to final assembly. Another important production measurement is total cost of unassembled raw goods. But many other unimportant pieces of information accompany the manufacturing information, such as actual date of production, shipment specification, and temperature at production.

To manage the data, an experienced DSS analyst or industrial engineer needs to determine what are the most important units of data. Then those units are stored in an online, easy-to-get-to location. There is efficiency of storage and efficiency of access. The remaining, less important detail is not discarded but is placed in a bulk storage location. In such a manner, large amounts of unstructured data can be efficiently stored and managed.

Modeling and External/Unstructured Data

What is the role of the data model and external data? Figure 8.6 reflects the dilemma. The normal role of the data model is the shaping of the environment, usually in terms of design. But external data and unstructured data are not malleable to any extent at all. Therefore, it appears that there is very little relationship between the data model and external data. About the best that can be done is to note the differences between the data model and external data as far as the interpretation of key phrases and words are concerned. Attempting to use the data model for any serious reshaping of the external or unstructured data is a mistake. The most that can be done is to create subsets of the data that are compatible with the existing internal data.

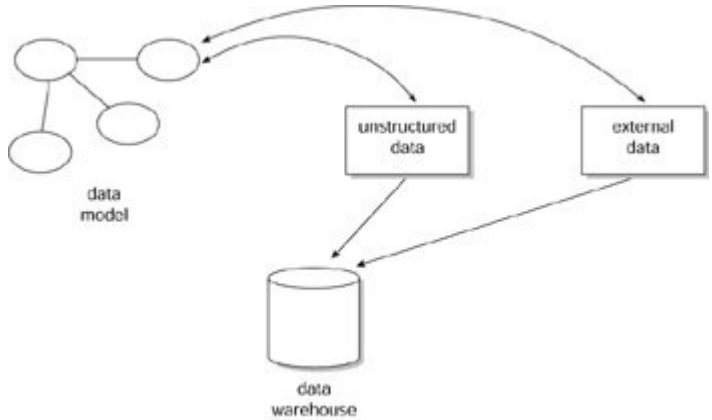

Figure 8.6: There is only a faint resemblance of external data/unstructured data to a

data model. Furthermore, nothing can be done about reshaping external data and unstructured data.

Secondary Reports

Not only can primary data be put in the data warehouse, but when data is repetitive in nature, secondary reports can be created from the detailed data over time. For example, take the month-end Dow Jones average report shown in Figure 8.7.

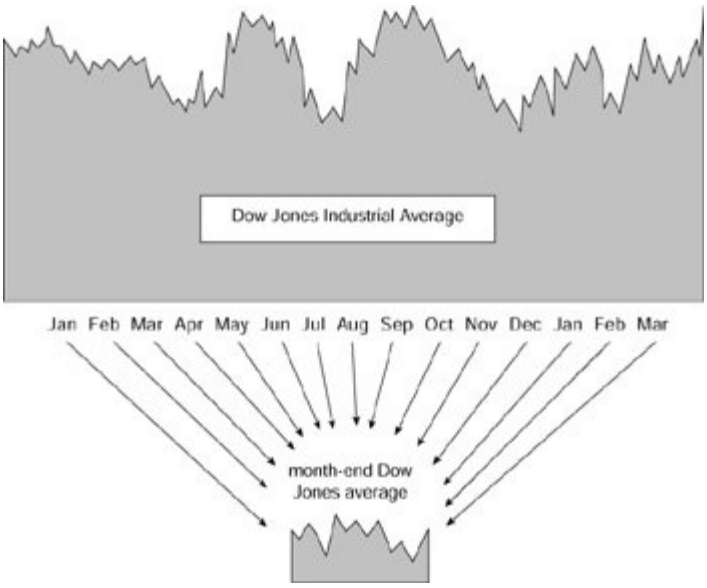

Figure 8.7: Creating a summary report from daily or monthly recurring information.

In the figure, Dow Jones information comes into the data warehouse environment daily. The daily information is useful, but of even more interest are the long-term trends that are formed. At the end of the month, the Dow Jones average is shuffled off into a secondary report. The secondary report then becomes part of the store of external data contained in the data warehouse.

Archiving External Data

Every piece of information—external or otherwise—has a useful lifetime. Once past that lifetime, it is not economical to keep the information. An essential part of managing external data is deciding what the useful lifetime of the data is. Even after this is determined, there remains the issue of whether the data should be discarded or put into archives. As a rule, external data may be removed from the data warehouse and placed on less expensive storage. The meta data reference to the external data is updated to reflect the new storage place and is left in the meta data store. The cost of an entry into the meta data store is so low that once put there, it is best left there.

Comparing Internal Data to External Data

One of the most useful things to do with external data is to compare it to internal data over a period of time. The comparison allows management a unique perspective. For instance, being able to contrast immediate and personal activities and trends against global activities

and trends allows an executive to have insights that simply are not possible elsewhere. Figure 8.8 shows such a comparison.

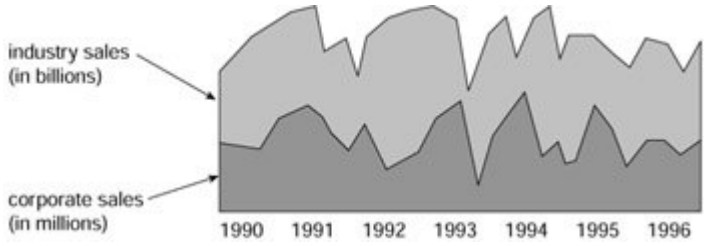

Figure 8.8: External data compared to internal data can be very elucidating.

When the comparison between external and internal data is made, the assumption is that the comparison is made on a common key. Any other assumption and the comparison between external and internal data loses much of its usefulness. Unfortunately, actually achieving a common-key basis between external and internal data is not easy.

To understand the difficulty, consider two cases. In one case, the commodity being sold is a large, expensive item, such as a car or a television set. For a meaningful comparison, sale by actual outlet needs to be measured. The actual sales by dealer is the basis for comparison. Unfortunately, the key structure used for dealers by the external source of data is not the same key structure used by internal systems. Either the external source must be converted to the key structure of the internal source or vice versa. Such a conversion is a nontrivial task.

Now consider the measurement of sales of a high-volume, low-cost item such as colas. The internal sales figures of the company reflect the sale of colas. But the external sales data has mixed the sales of colas with the sales of other beverages such as beer. Making a comparison between the two types of sales data will lead to some very misleading conclusions. For a meaningful comparison, there needs to be a “cleansing” of the external sales data to include only colas. In fact, if at all possible, colas only of the variety produced and sold by the bottler should be included. Not only should beer be removed from the external sales data, but noncompeting cola types should be removed as well.

Summary

The data warehouse is capable of holding much more than internal, structured data. There is much information relevant to the running of the company that comes from sources outside the company.

External data is captured, and information about the meta data is stored in the data warehouse meta data. External data often undergoes significant editing and transformation as the data is moved from the external environment to the data warehouse environment. The meta data that describes the external data and the unstructured data serves as an executive index to information. Much can be done with the index information, such as placing it in both disk storage and near-line storage, creating a link between the data warehouse and unstructured data, doing internal index processing, and so forth. In addition, a “notification” service is often provided whenever a new entry is made into the data warehouse.

External and unstructured data may or may not actually be stored in the data warehouse. By associating external and unstructured data with a data warehouse, the organization precludes the need to store the external and unstructured data in multiple places. Because of the bulk of data that is associated with unstructured data, it is best to store at least part of the unstructured data on a bulk storage medium such as near-line storage.

Chapter 9: Migration to the Architected Environment

Overview

Any architecture that must be implemented all at once, in a big bang, is doomed to failure in today's world. There simply is too much risk and too long a period to wait until there is a payback. In addition, trying to freeze changes to consider any path that is revolutionary, rather than evolutionary is unrealistic.

It is very good news indeed that migrating to the architected data warehouse environment is a step-by-step activity that is accomplished one finite deliverable at a time. The most successful implementations of the architected environment have been those in which the data warehouse has been built one iteration at a time. In doing so, the data warehouse can be built for a minimum of employee resources and with absolutely minimal disruption to the existing applications environment. Both the size and the speed of iterative development are important. Results must be delivered quickly.

In this chapter, a generic migration plan and a methodology for development are discussed. The methodology is described in detail in the appendix. The migration plan has been successfully followed by a wide number of companies; it is anything but a fanciful flight. The approach has been gleaned from the experiences of a number of companies. Of course, each company will have its own diversions and permutations. But the migration plan and approach have met with enough success in enough diverse companies to merit the confidence of the general-purpose developer.

A Migration Plan

The beginning point for the migration plan is a corporate data model. This model represents the information needs of the corporation. Keep in mind that it represents what the corporation needs, not necessarily what it currently has. In addition, it is built with no consideration of technology.

The corporate data model may be built internally, or it may have been generated from a generic data model. The corporate data model needs to identify (at a minimum!) the following:

- Major subjects of the corporation
- Definition of the major subjects of the corporation
- Relationships between the major subjects
- Groupings of keys and attributes that more fully represent the major subjects, including the following:
  - Attributes of the major subjects
  - Keys of the major subjects
  - Repeating groups of keys and attributes
- Connectors between major subject areas
- Subtyping relationships

In theory, it is possible to build the architected data-warehouse-centric environment without a data model; however, in practice, it is never done. Trying to build such an environment without a data model is analogous to trying to navigate without a map. It can be done, but like a person who has never been outside of Texas landing at New York's La Guardia airport and driving to midtown Manhattan with no map or instructions, it is very prone to trial and error.

Figure 9.1 shows that building or otherwise acquiring a data model is the starting point for the migration process. As a rule, the corporate data model identifies corporate information at a high level. From the corporate data model a lower-level model is built. The lower-level model identifies details that have been glossed over by the corporate data model. This midlevel model is built from the subject areas that have been identified in the corporate data model,t one subject area at a time. It is not built on an all-at-once basis because such doing so takes too long.

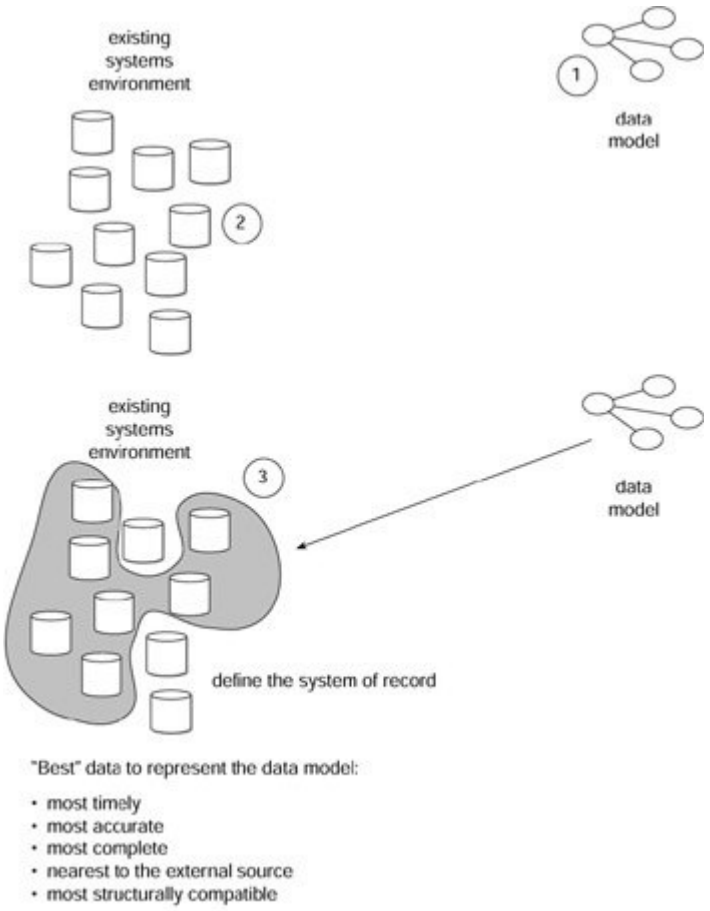

Figure 9.1: Migration to the architected environment.

Both the corporate data model and its associated midlevel model focus only on the atomic data of the corporation. No attempt is made to include derived data or DSS data in these models. Instead, derived data and DSS are deliberately excluded from the corporate data model and the midlevel models.

Some reasons for excluding derived data and DSS data from the corporate data model and the midlevel model include the following:

- Derived data and DSS data change frequently.
- These forms of data are created from atomic data.
- They frequently are deleted altogether.
- There are many variations in the creation of derived data and DSS data.

Because derived data and DSS data are excluded from the corporate data model and the midlevel model, the data model does not take long to build.

After the corporate data model and the midlevel models are in place, the next activity is defining the system of record. The system of record is defined in terms of the corporation's existing systems. Usually, these older legacy systems are affectionately known as the "mess."

The system of record is nothing more than the identification of the "best" data the corporation has that resides in the legacy operational or in the Web-based ebusiness environment. The data model is used as a benchmark for determining what the best data is. In other words, the data architect starts with the data model and asks what data is in hand that best fulfills the data requirements identified in the data model. It is understood that the fit will be less than perfect. In some cases, there will be no data in the existing systems environment or the Web-based ebusiness environment that exemplifies the data in the data model. In other cases, many sources of data in the existing systems environment contribute data to the systems of record, each under different circumstances.

The "best" source of existing data or data found in the Web-based ebusiness environment is determined by the following criteria:

- What data in the existing systems or Web-based ebusiness environment is the most complete?
- What data in the existing systems or Web-based ebusiness environment is the most timely?
- What data in the existing systems or Web-based ebusiness environment is the most accurate?
- What data in the existing systems or Web-based ebusiness environment is the closest to the source of entry into the existing systems or Web-based ebusiness environment?
- What data in the existing systems or Web-based ebusiness environment conforms the most closely to the structure of the data model? In terms of keys? In terms of attributes? In terms of groupings of data attributes?

Using the data model and the criteria described here, the analyst defines the system of record. The system of record then becomes the definition of the source data for the data warehouse environment. Once this is defined, the designer then asks what are the technological challenges in bringing the system-of-record data into the data warehouse. A short list of the technological challenges includes the following:

- A change in DBMS. The system of record is in one DBMS, and the data warehouse is in another DBMS.
- A change in operating systems. The system of record is in one operating system, and the data warehouse is in another operating system,
- The need to merge data from different DBMSs and operating systems. The system of record spans more than one DBMS and/or operating system. System-of-record data must be pulled from multiple DBMSs and multiple operating systems and must be merged in a meaningful way.
- The capture of the Web-based data in the Web logs. Once captured, how can the data be freed for use within the data warehouse?
- A change in basic data formats. Data in one environment is stored in ASCII, and data in the data warehouse is stored in EBCDIC, and so forth.

Another important technological issue that sometimes must be addressed is the volume of data. In some cases, huge volumes of data will be generated in the legacy environment. Specialized techniques may be needed to enter them into the data warehouse. For example, clickstream data found in the Web logs needs to be preprocessed before it can be used effectively in the data warehouse environment.

There are other issues. In some cases, the data flowing into the data warehouse must be cleansed. In other cases, the data must be summarized. A host of issues relate to the

mechanics of the bringing of data from the legacy environment into the data warehouse environment.

After the system of record is defined and the technological challenges in bringing the data into the data warehouse are identified, the next step is to design the data warehouse, as shown in Figure 9.2.

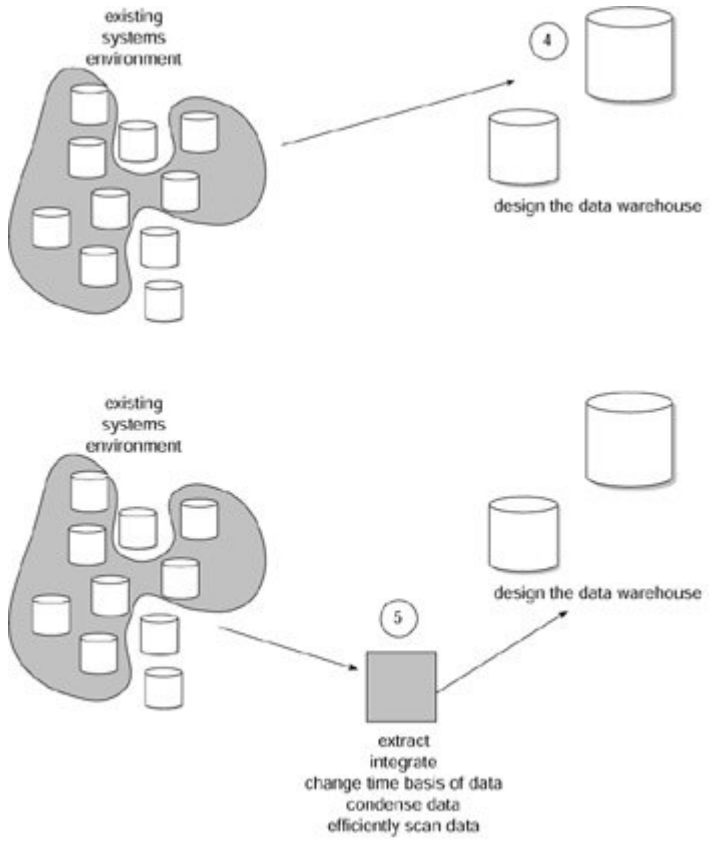

Figure 9.2: Migration to the architected environment.

If the data modeling activity has been done properly, the design of the data warehouse is fairly simple. Only a few elements of the corporate data model and the midlevel model need to be changed to turn the data model into a data warehouse design. Principally, the following needs to be done:

- An element of time needs to be added to the key structure if one is not already present.
- All purely operational data needs to be eliminated.
- Referential integrity relationships need to be turned into artifacts.
- Derived data that is frequently needed is added to the design.

The structure of the data needs to be altered when appropriate for the following:

- Adding arrays of data
- Adding data redundantly
- Further separating data under the right conditions
- Merging tables when appropriate

Stability analysis of the data needs to be done. In stability analysis, data whose content has a propensity for change is isolated from data whose content is very stable. For example, a

bank account balance usually changes its content very frequently-as much as three or four times a day. But a customer address changes very slowly-every three or four years or so. Because of the very disparate stability of bank account balance and customer address, these elements of data need to be separated into different physical constructs.

The data warehouse, once designed, is organized by subject area. Typical subject areas are as follows:

- Customer
- Product
- Sale
- Account
- Activity
- Shipment

Within the subject area there will be many separate tables, each of which is connected by a common key. All the customer tables will have CUSTOMER as a key, for example.

One of the important considerations made at this point in the design of the data warehouse is the number of occurrences of data. Data that will have very many occurrences will have a different set of design considerations than data that has very few occurrences. Typically, data that is voluminous will be summarized, aggregated, or partitioned (or all of the above). Sometimes profile records are created for voluminous data occurrences.

In the same vein, data that arrives at the data warehouse quickly (which is usually, but not always, associated with data that is voluminous) must be considered as well. In some cases, the arrival rate of data is such that special considerations must be made to handle the influx of data. Typical design considerations include staging the data, parallelization of the load stream, delayed indexing, and so forth.

After the data warehouse is designed, the next step is to design and build the interfaces between the system of record-in the operational environment-and the data warehouses. The interfaces populate the data warehouse on a regular basis.

At first glance, the interfaces appear to be merely an extract process, and it is true that extract processing does occur. But many more activities occur at the point of interface as well:

- Integration of data from the operational, application-oriented environment
- Alteration of the time basis of data
- Condensation of data
- Efficient scanning of the existing systems environment

Most of these issues have been discussed elsewhere in this book.

Note that the vast majority of development resources required to build a data warehouse are consumed at this point. It is not unusual for 80 percent of the effort required to build a data warehouse to be spent here. In laying out the development activities for building a data warehouse, most developers overestimate the time required for other activities and underestimate the time required for designing and building the operational-to-data-warehouse interface. In addition to requiring resources for the initial building of the interface into the data warehouse, the ongoing maintenance of the interfaces must be considered. Fortunately, ETL software is available to help build and maintain this interface.

Once the interface programs are designed and built, the next activity is to start the population of the first subject area, as shown in Figure 9.3. The population is conceptually very simple. The first of the data is read in the legacy environment; then it is captured and transported to the data warehouse environment. Once in the data warehouse environment

the data is loaded, directories are updated, meta data is created, and indexes are made. The first iteration of the data is now ready for analysis in the data warehouse.

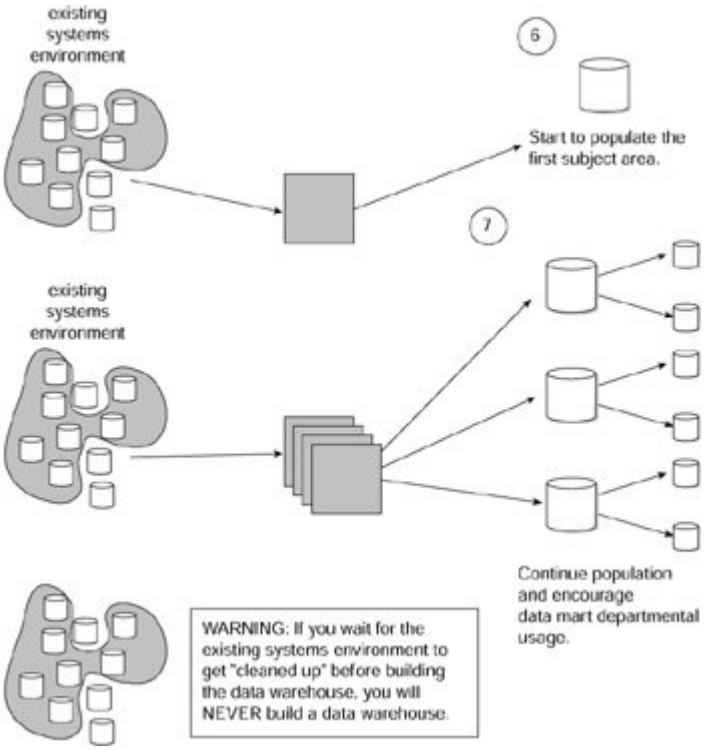

Figure 9.3: Iterative migration to the architected environment.

There are many good reasons to populate only a fraction of the data needed in a data warehouse at this point. Changes to the data likely will need to be made. Populating only a small amount of data means that changes can be made easily and quickly. Populating a large amount of data greatly diminishes the flexibility of the data warehouse. Once the end user has had a chance to look at the data (even just a sample of the data) and give feedback to the data architect, then it is safe to populate large volumes of data. But before the end user has a chance to experiment with the data and to probe it, it is not safe to populate large volumes of data.

End users operate in a mode that can be called the “discovery mode.” End users don’t know what their requirements are until they see what the possibilities are. Initially populating large amounts of data into the data warehouse is dangerous-it is a sure thing that the data will change once populated. Jon Geiger says that the mode of building the data warehouse is “build it wrong the first time.” This tongue-in-cheek assessment has a strong element of truth in it.

The population and feedback processes continue for a long period (indefinitely). In addition, the data in the warehouse continues to be changed. Of course, over time, as the data becomes stable, it changes less and less.

A word of caution: If you wait for existing systems to be cleaned up, you will never build a data warehouse. The issues and activities of the existing systems’ operational environment must be independent of the issues and activities of the data warehouse environment. One train of thought says, “Don’t build the data warehouse until the operational environment is cleaned up.” This way of thinking may be theoretically appealing, but in truth it is not practical at all.

One observation worthwhile at this point relates to the frequency of refreshment of data into the data warehouse. As a rule, data warehouse data should be refreshed no more frequently than every 24 hours. By making sure that there is at least a 24-hour time delay in the loading of data, the data warehouse developer minimizes the temptation to turn the data warehouse into an operational environment. By strictly enforcing this lag of time, the data warehouse serves the DSS needs of the company, not the operational needs. Most operational processing depends on data being accurate as of the moment of access (i.e., current-value data). By ensuring that there is a 24-hour delay (at the least), the data warehouse developer adds an important ingredient that maximizes the chances for success.

In some cases, the lag of time can be much longer than 24 hours. If the data is not needed in the environment beyond the data warehouse, then it may make sense not to move the data into the data warehouse on a weekly, monthly, or even quarterly basis. Letting the data sit in the operational environment allows it to settle. If adjustments need to be made, then they can be made there with no impact on the data warehouse if the data has not already been moved to the warehouse environment.

The Feedback Loop

At the heart of success in the long-term development of the data warehouse is the feedback loop between the data architect and the DSS analyst, shown in Figure 9.4. Here the data warehouse is populated from existing systems. The DSS analyst uses the data warehouse as a basis for analysis. On finding new opportunities, the DSS analyst conveys those requirements to the data architect, who makes the appropriate adjustments. The data architect may add data, delete data, alter data, and so forth based on the recommendations of the end user who has touched the data warehouse.

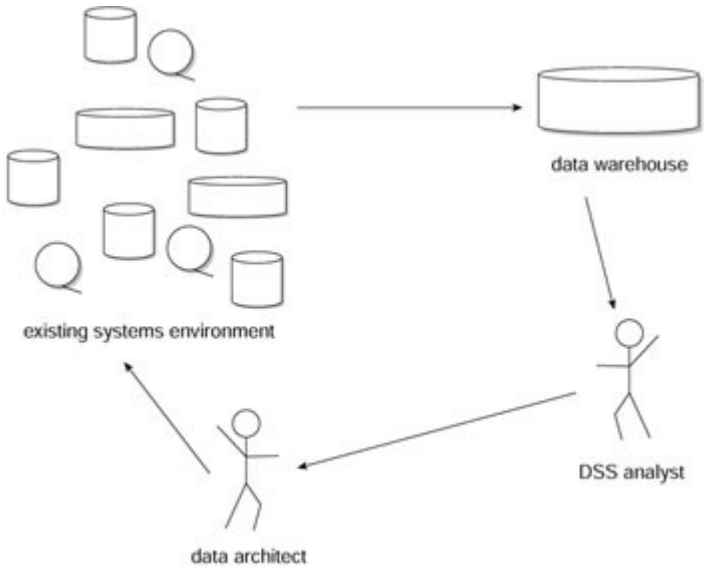

Figure 9.4: The crucial feedback loop between DSS analyst and data architect.

A few observations about this feedback loop are of vital importance to the success of the data warehouse environment:

- The DSS analyst operates—quite legitimately—in a “give me what I want, then I can tell you what I really want” mode. Trying to get requirements from the DSS analyst before he or she knows what the possibilities are is an impossibility.

- The shorter the cycle of the feedback loop, the more successful the warehouse effort. Once the DSS analyst makes a good case for changes to the data warehouse, those changes need to be implemented as soon as possible.
- The larger the volume of data that has to be changed, the longer the feedback loop takes. It is much easier to change 10 gigabytes of data than 100 gigabytes of data.

Failing to implement the feedback loop greatly short-circuits the probability of success in the data warehouse environment.

Strategic Considerations

Figure 9.5 shows that the path of activities that have been described addresses the DSS needs of the organization. The data warehouse environment is designed and built for the purpose of supporting the DSS needs of the organization, but there are needs other than DSS needs.

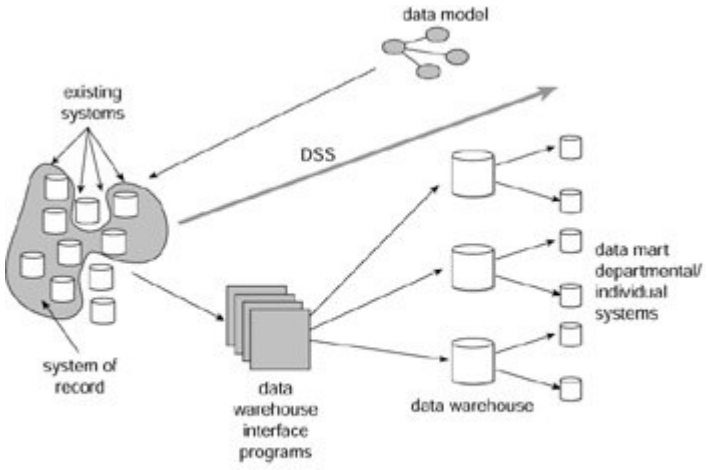

Figure 9.5: The first major path to be followed is DSS.

Figure 9.6 shows that the corporation has operational needs as well. In addition, the data warehouse sits at the hub of many other architectural entities, each of which depends on the data warehouse for data.

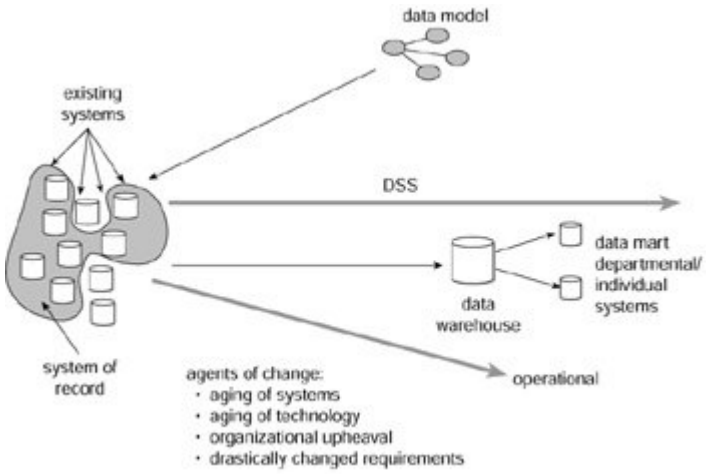

Figure 9.6: To be successful, the data architect should wait for agents of change to become compelling and ally the efforts toward the architected environment with the appropriate agents.

In Figure 9.6, the operational world is shown as being in a state of chaos. There is much unintegrated data and the data and systems are so old and so patched they cannot be maintained. In addition, the requirements that originally shaped the operational applications have changed into an almost unrecognizable form.

The migration plan that has been discussed is solely for the construction of the data warehouse. Isn't there an opportunity to rectify some or much of the operational "mess" at the same time that the data warehouse is being built? The answer is that, to some extent, the migration plan that has been described presents an opportunity to rebuild at least some of the less than aesthetically pleasing aspects of the operational environment.

One approach—which is on a track independent of the migration to the data warehouse environment—is to use the data model as a guideline and make a case to management that major changes need to be made to the operational environment. The industry track record of this approach is dismal. The amount of effort, the amount of resources, and the disruption to the end user in undertaking a massive rewrite and restructuring of operational data and systems is such that management seldom supports such an effort with the needed level of commitment and resources.

A better ploy is to coordinate the effort to rebuild operational systems with what are termed the "agents of change":

- The aging of systems
- The radical changing of technology
- Organizational upheaval
- Massive business changes

When management faces the effects of the agents of change, there is no question that changes will have to be made—the only question is how soon and at what expense. The data architect allies the agents of change with the notion of an architecture and presents management with an irresistible argument for the purpose of restructuring operational processing.

The steps the data architect takes to restructure the operational environment—which is an activity independent of the building of the data warehouse—are shown in Figure 9.7.

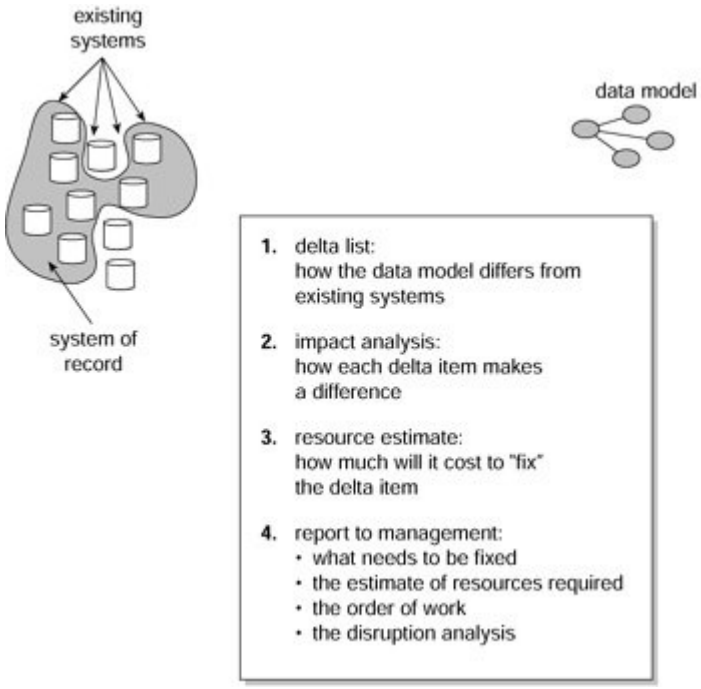

Figure 9.7: The first steps in creating the operational cleanup plan.

First a “delta” list is created. The delta list is an assessment of the differences between the operational environment and the environment depicted by the data model. The delta list is simple, with very little elaboration.

The next step is the impact analysis. At this point an assessment is made of the impact of each item on the delta list. Some items may have a serious impact; other items may have a negligible impact on the running of the company.

Next, the resource estimate is created. This estimate is for the determination of how many resources will be required to “fix” the delta list item.

Finally, all the preceding are packaged in a report that goes to information systems management. Management then makes a decision as to what work should proceed, at what pace, and so forth. The decision is made in light of all the priorities of the corporation.

Methodology and Migration

In the appendix of this book, a methodology for building a data warehouse is described. The methodology is actually a much larger one in scope in that it not only contains information about how to build a data warehouse but also describes how to use the data warehouse. In addition, the classical activities of operational development are included to form what can be termed a data-driven methodology.

The methodology described differs from the migration path in several ways. The migration path describes general activities dynamically. The methodology describes specific activities, deliverables from those activities, and the order of the activities. The iterative dynamics of creating a warehouse are not described, though. In other words, the migration plan describes a sketchy plan in three dimensions, while the methodology describes a detailed plan in one dimension. Together they form a complete picture of what is required to build the data warehouse.

A Data-Driven Development Methodology

Development methodologies are quite appealing to the intellect. After all, methodology directs the developer down a rational path, pointing out what needs to be done, in what order, and how long the activity should take. However, as attractive as the notion of a methodology is, the industry track record has not been good. Across the board, the enthusiasm for methodologies (data warehouse or any other) has met with disappointment on implementation.

Why have methodologies been disappointing? The reasons are many:

- Methodologies generally show a flat, linear flow of activities. In fact, almost any methodology requires execution in terms of iterations. In other words, it is absolutely normal to execute two or three steps, stop, and repeat all or part of those steps again. Methodologies usually don't recognize the need to revisit one or more activities. In the case of the data warehouse, this lack of support for iterations makes a methodology a very questionable subject.
- Methodologies usually show activities as occurring once and only once. Indeed, while some activities need to be done (successfully) only once, others are done repeatedly for different cases (which is a different case than reiteration for refinement).
- Methodologies usually describe a prescribed set of activities to be done. Often, some of the activities don't need to be done at all, other activities need to be done that are not shown as part of the methodology, and so forth.
- Methodologies often tell how to do something, not what needs to be done. In describing how to do something, the effectiveness of the methodology becomes mired in detail and in special cases.
- Methodologies often do not distinguish between the sizes of the systems being developed under the methodology. Some systems are so small that a rigorous methodology makes no sense. Some systems are just the right size for a methodology. Other systems are so large that their sheer size and complexity will overwhelm the methodology.
- Methodologies often mix project management concerns with design/development activities to be done. Usually, project management activities should be kept separate from methodological concerns.
- Methodologies often do not make the distinction between operational and DSS processing. The system development life cycles for operational and DSS processing are diametrically opposed in many ways. A methodology must distinguish between operational and DSS processing and development in order to be successful.
- Methodologies often do not include checkpoints and stopping places in the case of failure. "What is the next step if the previous step has not been done properly?" is usually not a standard part of a methodology.
- Methodologies are often sold as solutions, not tools. When a methodology is sold as a solution, inevitably it is asked to replace good judgment and common sense, and this is always a mistake.
- Methodologies often generate a lot of paper and very little design. Design and development activities are not legitimately replaced by paper.

Methodologies can be very complex, anticipating every possibility that may ever happen. Despite these drawbacks, there still is some general appeal for methodologies. A general-purpose methodology—applicable to the data-driven environment—is described in the appendix, with full recognition of the pitfalls and track record of methodologies. The data-driven methodology that is outlined owes much to its early predecessors. As such, for a much fuller explanation of the intricacies and techniques described in the methodology, refer to the books listed in the references in the back of this book.

One of the salient aspects of a data-driven methodology is that it builds on previous efforts—utilizing both code and processes that have already been developed. The only way that

development on previous efforts can be achieved is through the recognition of commonality. Before the developer strikes the first line of code or designs the first database, he or she needs to know what already exists and how it affects the development process. A conscious effort must be made to use what is already in place and not reinvent the wheel. That is one of the essences of data-driven development.

The data warehouse environment is built under what is best termed an iterative development approach. In this approach a small part of the system is built to completion, then another small part is completed, and so forth. That development proceeds down the same path repeatedly makes the approach appear to be constantly recycling itself. The constant recycling leads to the term “spiral” development.

The spiral approach to development is distinct from the classical approach, which can be called the “waterfall” approach. In this approach all of one activity is completed before the next activity can begin, and the results of one activity feed another. Requirements gathering is done to completion before analysis and synthesization commence. Analysis and synthesization are done to completion before design begins. The results of analysis and synthesization feed the process of design, and so forth. The net result of the waterfall approach is that huge amounts of time are spent making any one step complete, causing the development process to move at a glacial speed.

Figure 9.8 shows the differences between the waterfall approach and the spiral approach.

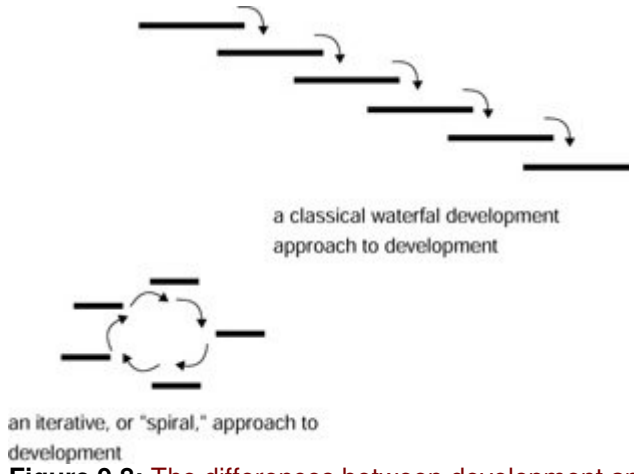

Figure 9.8: The differences between development approaches, from a high level.

Because the spiral development process is driven by a data model, it is often said to be data driven.

Data-Driven Methodology

What makes a methodology data driven? How is a data-driven methodology any different from any other methodology? There are at least two distinguishing characteristics of a data-driven methodology.

A data-driven methodology does not take an application-by-application approach to the development of systems. Instead, code and data that have been built previously are built on, rather than built around. To build on previous efforts, the commonality of data and processing must be recognized. Once recognized, data is built on if it already exists; if no

data exists, data is constructed so that future development may built on it. The key to the recognition of commonality is the data model.

There is an emphasis on the central store of data—the data warehouse—as the basis for DSS processing, recognizing that DSS processing has a very different development life cycle than operational systems.

System Development Life Cycles

Fundamentally, shaping the data-driven development methodology is the profound difference in the system development life cycles of operational and DSS systems. Operational development is shaped around a development life cycle that begins with requirements and ends with code. DSS processing begins with data and ends with requirements.

A Philosophical Observation

In some regards, the best example of methodology is the Boy Scout and Girl Scout merit badge system, which is used to determine when a scout is ready to pass to the next rank. It applies to both country- and city-dwelling boys and girls, the athletically inclined and the intellectually inclined, and to all geographical areas. In short, the merit badge system is a uniform methodology for the measurement of accomplishment that has stood the test of time.

Is there is any secret to the merit badge methodology? If so, it is this: The merit badge methodology does not prescribe how any activity is to be accomplished; instead, it merely describes what is to be done with parameters for the measurement of the achievement. The how-to that is required is left up to the Boy Scout or Girl Scout.

Philosophically, the approach to methodology described in the appendix of this book takes the same perspective as the merit badge system. The results of what must be accomplished and, generally speaking, the order in which things must be done is described. How the results required are to be achieved is left entirely up to the developer.

Operational Development/DSS Development

The data-driven methodology will be presented in three parts: METH 1, METH 2, and METH 3. The first part of the methodology, METH 1, is for operational systems and processing. This part of the methodology will probably be most familiar to those used to looking at classically structured operational methodologies. METH 2 is for DSS systems and processing—the data warehouse. The essence of this component of the methodology is a data model as the vehicle that allows the commonality of data to be recognized. It is in this section of the appendix that the development of the data warehouse is described. The third part of the methodology, METH 3, describes what occurs in the heuristic component of the development process. It is in METH 3 that the usage of the warehouse is described.

Summary

In this chapter, a migration plan and a methodology (found in the appendix) were described. The migration plan addresses the issues of transforming data out of the existing systems environment into the data warehouse environment. In addition, the dynamics of how the operational environment might be organized were discussed.

The data warehouse is built iteratively. It is a mistake to build and populate major portions of the data warehouse—especially at the beginning—because the end user operates in what can be termed the “mode of discovery.” The end user cannot articulate what he or she wants until the possibilities are known.

The process of integration and transformation of data typically consumes up to 80 percent of development resources. In recent years, ETL software has automated the legacy-to-data-warehouse interface development process.

The starting point for the design of the data warehouse is the corporate data model, which identifies the major subject areas of the corporation. From the corporate data model is created a lower-level “midlevel model.” The corporate data model and the midlevel model are used as a basis for database design. After the corporate data model and the midlevel model have been created, such factors as the number of occurrences of data, the rate at which the data is used, the patterns of usage of the data, and more are factored into the design.

The development approach for the data warehouse environment is said to be an iterative or a spiral development approach. The spiral development approach is fundamentally different from the classical waterfall development approach.

A general-purpose, data-driven methodology was also discussed. The general-purpose methodology has three phases—an operational phase, a data warehouse construction phase, and a data warehouse iterative usage phase.

The feedback loop between the data architect and the end user is an important part of the migration process. Once the first of the data is populated into the data warehouse, the data architect listens very carefully to the end user, making adjustments to the data that has been populated. This means that the data warehouse is in constant repair. During the early stages of the development, repairs to the data warehouse are considerable. But as time passes and as the data warehouse becomes stable, the number of repairs drop off.

Chapter 10: The Data Warehouse and the Web

Overview

One of the most widely discussed technologies is the Internet and its associated environment-the World Wide Web. Embraced by Wall Street as the basis for the new economy, Web technology enjoys wide popular support among business people and technicians alike. Although not obvious at first glance, there is a very strong affinity between the Web sites built by organizations and the data warehouse. Indeed, data warehousing provides the foundation for the successful operation of a Web-based ebusiness environment.

The Web environment is owned and managed by the corporation. In some cases, the Web environment is outsourced. But in most cases the Web is a normal part of computer operations, and it is often used as a hub for the integration of business systems. (Note that if the Web environment is outsourced, it becomes much more difficult to capture, retrieve, and integrate Web data with corporate processing.)

The Web environment interacts with corporate systems in two basic ways. One interaction occurs when the Web environment creates a transaction that needs to be executed-an order from a customer, for example. The transaction is formatted and shipped to corporate systems, where it is processed just like any other order. In this regard, the Web is merely another source for transactions entering the business.

But the Web interacts with corporate systems another way as well—through the collection of Web activity in a log. Figure 10.1 shows the capture of Web activity and the placement of that activity in a log.

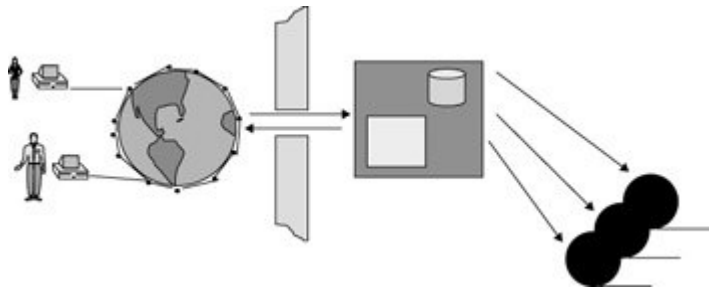

Figure 10.1: The activity of the Web environment is spun off into Web logs in records called clickstream records.

The Web log contains what is typically called clickstream data. Each time the Internet user clicks to move to a different location, a clickstream record is created. As the user looks at different corporate products, a record of what the user has looked at, what the user has purchased, and what the user has thought about purchasing is compiled. Equally important, what the Internet user has not looked at and has not purchased can be determined. In a word, the clickstream data is the key to understanding the stream of consciousness of the Internet user. By understanding the mindset of the Internet user, the business analyst can understand very directly how products, advertising, and promotions are being received by the public, in a way much more quantified and much more powerful than ever before.

But the technology required to make this powerful interaction happen is not trivial. There are some obstacles to understanding the data that comes from the Web environment. For example, Web-generated data is at a very low level of detail-in fact, so low that it is not fit for

either analysis or entry into the data warehouse. To make the clickstream data useful for analysis and the warehouse, the log data must be read and refined.

Figure 10.2 shows that Web log clickstream data is passed through software that is called a *Granularity Manager* before entry into the data warehouse environment.

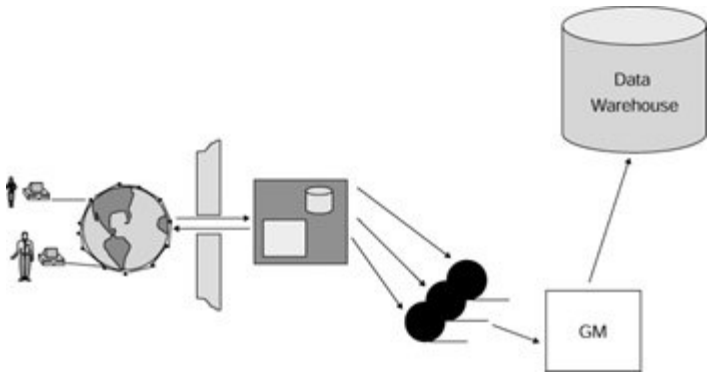

Figure 10.2: Data passes through the Granularity Manager before entering the data warehouse.

A lot of processing occurs in the Granularity Manager, which reads clickstream data and does the following:

- Edits out extraneous data
- Creates a single record out of multiple, related clickstream log records
- Edits out incorrect data
- Converts data that is unique to the Web environment, especially key data that needs to be used in the integration with other corporate data
- Summarizes data
- Aggregates data

As a rule of thumb, about 90 percent of raw clickstream data is discarded or summarized as it passes through the Granularity Manager. Once passed through the manager into the data warehouse, the clickstream data is ready for integration into the mainstream of corporate processing.

In summary, the process of moving data from the Web into the data warehouse involves these steps:

- Web data is collected into a log.
- The log data is processed by passing through a Granularity Manager.
- The Granularity Manager then passes the refined data into the data warehouse.

The way that data passes back into the Web environment is not quite as straightforward. Simply stated, the data warehouse does not pass data directly back into the Web environment. To understand why there is a less-than-straightforward access of data warehouse data, it is important to understand why the Web environment needs data warehouse data in the first place.

The Web environment needs this type of data because it is in the data warehouse that corporate information is integrated. For example, suppose there's a Web site dedicated to selling clothes. Now suppose the business analyst decides that it would be nice for a clothing customer become a customer for other goods the business sells, such as gardening tools, sports gear, travel accessories, and costume jewelry. The analyst might decide to initiate a special promotion for fancy women's dresses and upscale costume jewelry. But where does the analyst turn to find which women customers have bought costume jewelry in

the past? Why, naturally, he or she turns to the data warehouse because that is where the historical information about customers is found.

In another example, suppose the Web site is dedicated to selling cars. The analyst would really like to know who has purchased the brand of car the company is selling. Where is the historical information of this variety found? In the data warehouse, of course.

The data warehouse then provides a foundation of integrated historical information that is available to the business analyst. This affinity between the data warehouse and the Web is shown in Figure 10.3.

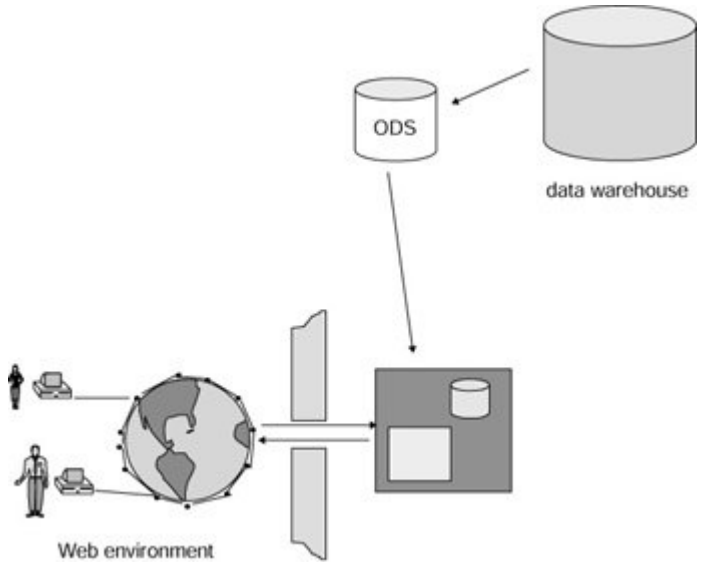

Figure 10.3: Data is passed to the ODS before it goes to the Web.

Figure 10.3 shows that data passes out of the data warehouse into the corporate operational data store (ODS), where it is then available for direct access from the Web. At first glance, it may seem odd that the ODS sits between the data warehouse and the Web. There are some very good reasons for this positioning.

The ODS is a hybrid structure that has some aspects of a data warehouse and other aspects of an operational system. The ODS contains integrated data and can support DSS processing. But the ODS can also support high-performance transaction processing. It is this last characteristic of the ODS that makes it so valuable to the Web.

When a Web site accesses the ODS, the Web environment knows that it will receive a reply in a matter of milliseconds. This speedy response time makes it possible for the Web to perform true transaction processing. If the Web were to directly access the data warehouse, it could take minutes to receive a reply from the warehouse. In the world of the Internet, where users are highly sensitive to response time, this would be unacceptable. Clearly, the data warehouse is not designed to support online response time. However, the ODS is designed for that purpose. Therefore, the direct input into the Web environment is the ODS, as seen by Figure 10.4.

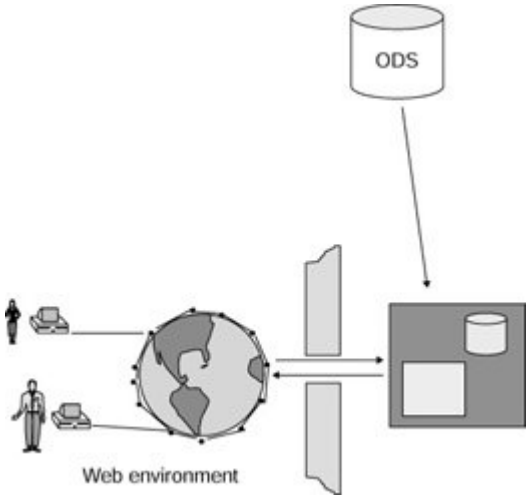

Figure 10.4: The ODS provides fast response time.

At first glance it may seem that there is a lot of redundant data between the data warehouse and the ODS. After all, the ODS is fed from the data warehouse. (Note: The ODS being discussed here is a class IV ODS. For a complete description of the other classes of ODS, refer to my book *Building the Operational Data Store*, Second Edition (Wiley, 1999).

But in truth there is very little overlap of data between the data warehouse and the ODS. The data warehouse contains detailed transaction data, while the ODS contains what can be termed “profile” data. To understand the differences be-tween profile data and detailed transaction data, consider the data seen in Figure 10.5.

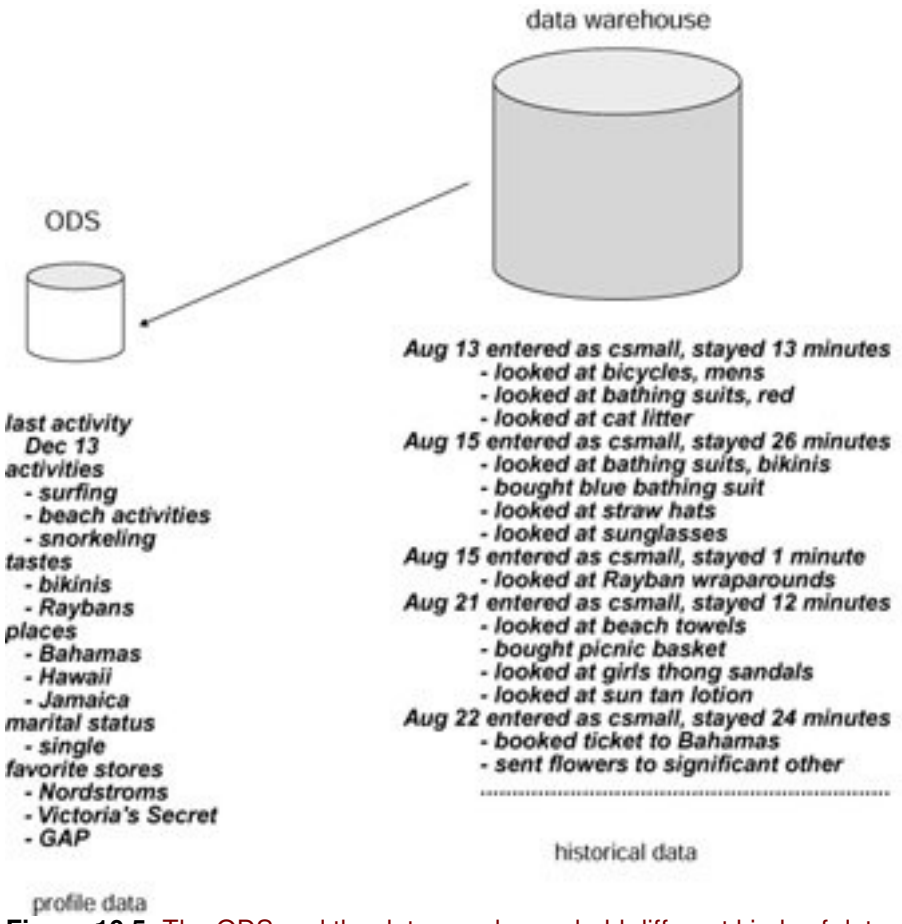

Figure 10.5: The ODS and the data warehouse hold different kinds of data.

The data warehouse contains all sorts of transaction data about past interactions between the customer and the business. Detailed transaction data includes information about the following:

- Searches for men's bicycles
- Searches for women's red bathing suits
- Purchases of a women's blue bathing suits
- Searches for Ray-Ban wraparounds

The data warehouse maintains a detailed log, by customer, of the transactional interactions the customer has had with the business, regardless of the source of the interaction. The indata warehouse. The profile record contains all sorts of information that is created as a result of reading and interpreting the transaction data. For example, for the customer shown for Figure 10.7, the profile record shows that the customer is all of the following:

- A beach-oriented person, interested in surfing, sun bathing, and snorkeling
- Likely to travel to places like the Bahamas, Hawaii, and Jamaica
- Single
- An upscale shopper who is likely to frequent places such as Nordstrom, Victoria's Secret, and the Gap

In other words, the customer is likely to have the propensities and proclivities shown in the profile record seen in Figure 10.7. Note that the customer may never have been to Hawaii. Nevertheless, it is predicted that the customer would like to go there.

To create the profile data from the transaction data, a certain amount of analysis must be done. Figure 10.6 shows the reading of the transactional data in order to produce profile data.

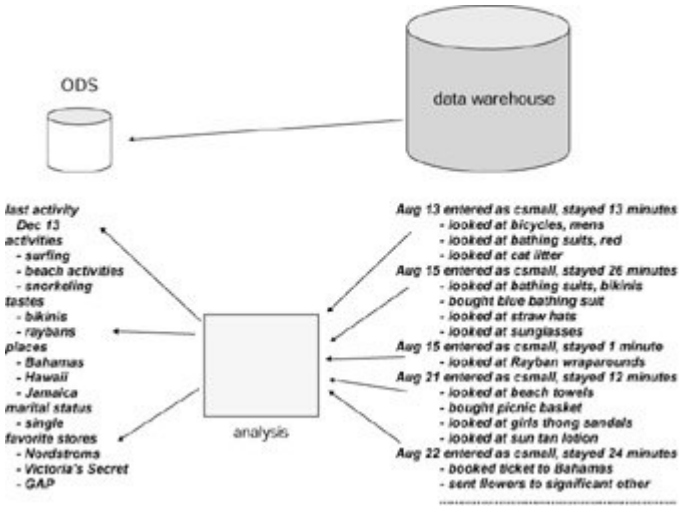

Figure 10.6: Periodically the detailed historical data is read, analyzed, and loaded into the format required for the ODS.

In Figure 10.6, the detailed integrated historical transaction data is read and analyzed in order to produce the profile record. The analysis is done periodically, depending on the rate of change of data and the business purpose behind the analysis. The frequency of analysis and subsequent update of the profile record may occur as often as once a day or as infrequently as once a year. There is wide variability in the frequency of analysis.

The analytical program is both interpretive and predictive. Based on the past activity of the customer and any other information that the analytical program can get, the analytical program assimilates the information to produce a very personal projection of the customer. The projection is predictive as well as factual. Certain factual information is standard:

- Date of the last interaction with the customer
- Nature of the last interaction
- Size of the last purchase

Other information is not nearly as factual. The predictive aspect of the analysis includes such information as the following:

- Whether the customer is upscale
- The customer's sex
- The customer's age
- Whether the customer is a frequent traveler
- Where the customer is likely to travel

The profile record then contains a thumbnail sketch of the customer that is available in the ODS in an instant. And in that instant, the Web environment is provided with excellent response time and an integrated, interpretive view of the customer base that is being served.

Of course, information other than customer information is available from the data warehouse and the ODS. Typically, vendor information, product information, sales information, and the like are also available for the Web analyst.

Providing good response time and preanalyzed data are not the only roles the data warehouse environment plays in supporting the Web. Another key role is the management of large amounts of data.

Web processing generates very large amounts of information. Even when a Granularity Manager is used to maximum effectiveness, the Web site still spews forth a mountain of data.

The first impulse of many Web designers is to store Web data in the Web environment itself. But very quickly the Web becomes swamped, and once that happens, nothing works properly. Data becomes entangled in everything-in access queries, in loads, in indexes, in monitors, and elsewhere. Coming to the aid of the Web is the data warehouse itself, as well as the bulk storage overflow component of the data warehouse. Figure 10.7 shows that data is periodically offloaded into the data warehouse from the Web environment. It is then periodically offloaded from the data warehouse to the overflow environment.

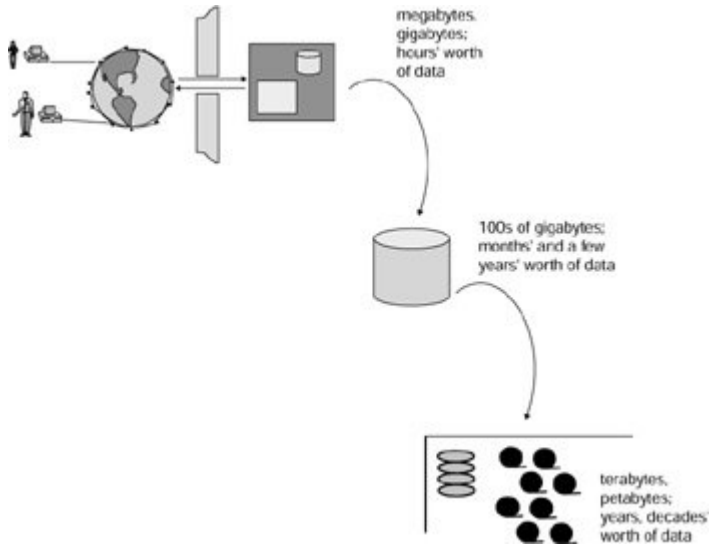

Figure 10.7: Volumes of data cascade down from the Web to the data warehouse to alternative storage.

The Granularity Manager takes care of loading data from the Web to the data warehouse on a daily or even an hourly basis, depending on the average amount of Web traffic. And data from the data warehouse is loaded monthly or quarterly to the overflow storage environment. In so doing, there never is an unmanageable amount of data at any level of the architecture.

Typically, the Web environment holds a day's worth of data, while the data warehouse might hold a year's worth. And typically the overflow storage component holds as much as a decade's worth of data. The data warehouse also supports the Web environment with the integration of data. Figure 10.8 shows that normal operational systems feed data to the data warehouse, where it becomes available for integrated processing. Data comes out of the Granularity Manager to merge with previously integrated business data in the data warehouse. In so doing, the data warehouse becomes the single place where you can get an integrated view of all business data from the Web, from other systems, from anywhere.

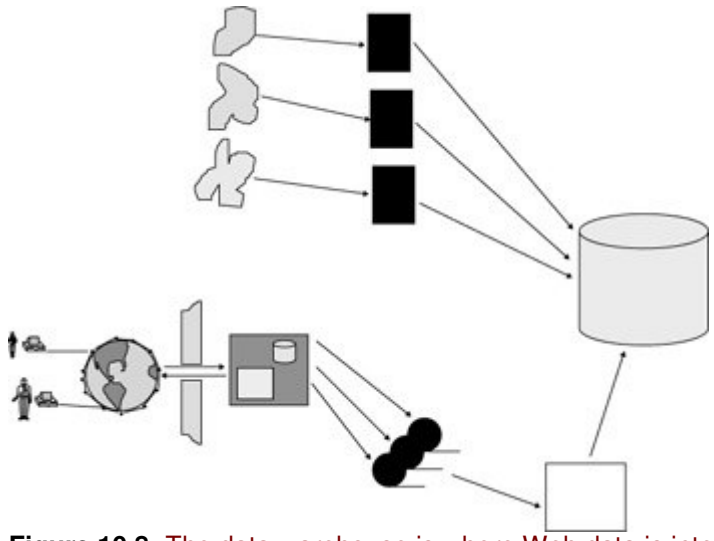

Figure 10.8: The data warehouse is where Web data is integrated with other data from the corporation.

Another important aspect of the data warehouse is its ability to support multiple Web sites. For a large corporation, multiple Web sites are a fact of life, and their support is essential for merging and integrating the data from all sites.

Supporting the Ebusiness Environment

A final environment that is supported by the data warehouse is the Web-based ebusiness environment. Figure 10.9 shows the support of the Web environment by the data warehouse.

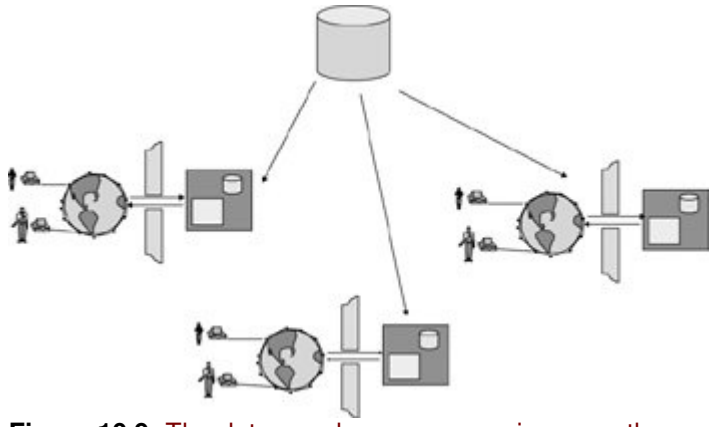

Figure 10.9: The data warehouse can service more than one ebusiness.

The interface between the Web environment and the data warehouse is at the same time both simple and complex. It is simple from the perspective that data moved from the data warehouse back and forth to the Web environment. It is complex in that the movement is anything less than straightforward.

Moving Data from the Web to the Data Warehouse

Data in the Web environment is collected at a very, very low level of detail—too low a level to be of use in the data warehouse. So, as the data passes from the Web environment to the data warehouse, it must be conditioned and its level of granularity must be raised. The sorts of things that are done to the data in the Web environment before becoming useful in the data warehouse are the following:

- Extraneous data is removed.
- Like occurrences of data are added together.
- Data is resequenced.
- Data is edited.
- Data is cleansed.
- Data is converted.

In short, the Web-based data goes through a rigorous cleansing/conversion/reduction exercise before it is fit for inclusion into the data warehouse.

The Web-based data usually comes through the Web logs that are created in the Web environment. There is, as a rule of thumb, about a 90 percent reduction of data that occurs as the data from the Web is reduced.

The data coming from the Web passes through software that is often called Granularity Management software. In many ways, the Granularity Management software is akin to the ETL software found in the movement of data from the legacy environment to the data warehouse.

The data coming into the Web environment comes primarily from the clickstream processing that occurs in the Web environment. Clickstream processing is good for telling what has happened in the Web-based user sessions. To be really useful, though, the clickstream data must be connected to the other mainstream data that passes through normal corporate systems. It is only when clickstream data has been distilled and merged with normal corporate data that the full benefit of Web information can be felt.

Moving Data from the Data Warehouse to the Web

The Web environment is very sensitive to response time; it cannot be kept waiting more than a millisecond or two when it needs information. If the Web environment must wait longer than that, performance will be impacted. In many regards, the Web environment is very similar to the OLTP environment, at least as far as response-time sensitivity is concerned. It is for these reasons that there is no direct interface between the data warehouse and the Web environment.

Instead, the interface between the two environments passes through the corporate ODS residing in the same environment as the data warehouse. The ODS is designed to provide millisecond response time; the data warehouse is not.

Therefore, data passes from the data warehouse to the ODS. Once in the ODS the data waits for requests for access from the Web environment. The Web then makes a request and gets the needed information very quickly and very consistently.

The ODS contains profile information, unlike the data warehouse, which contains detailed historical information. In addition, the ODS contains truly corporatewide information.

Once the data from the ODS passes into the Web environment, it can be used in any number of ways. The data can be used to shape the dialogues the Web has with its users,

for personalization, or for direct dialogue. In short, the data coming from the ODS/data warehouse can be used as the creativity of the Web designer demands.

Web Support

What exactly is it that the data warehouse provides for the Web-based ebusiness environment? The data warehouse provides several important capabilities:

- The ability to absorb huge amounts of data. Once the data warehouse is equipped with an overflow storage mechanism such as alternate/near-line storage and once the Web data passes through a Granularity Manager, the data warehouse is equipped to effectively handle an infinite amount of data. The data is quickly moved through the Web environment into the data warehouse. In doing so, the volumes of data generated by the Web environment are not an impediment to performance or availability in the Web environment.
- Access to integrated data. Web data by itself is naked and fairly useless. But once Web-generated data is combined with other corporate data, the mixture is very powerful. Web data once put into the data warehouse is able to be integrated, and in doing so, very useful information is created.
- The ability to provide very good performance. Because the Web accesses the ODS, not the data warehouse, good performance is consistently achieved.

These then are the important features that the data warehouse provides to the Web-based ebusiness environment. The data warehouse provides the important background infrastructure that the Web needs in order to be successful.

Summary

We have seen that the Web environment is supported by the data warehouse in a variety of ways. The interface for moving data from the Web to the data warehouse is fairly simple. Web data is trapped in logs. The logs feed their clickstream information to the Granularity Manager. The Granularity Manager edits, filters, summarizes, and reorganizes data. The data passes out of the Granularity Manager into the data warehouse.

The interface for moving data from the warehouse to the Web is a little more complex. Data passes from the data warehouse into an ODS. In the ODS a profile record is created. The ODS becomes the sole point of contact between the Web environment and the data warehouse for purposes of data flow from the data warehouse to the Web environment. The reason for this is simple: The ODS is able to ensure that online transactions are processed quickly and consistently, which is essential for efficient Web processing.

In addition, the data warehouse provides a place where massive amounts of data can be downloaded from the Web environment and stored.

The data warehouse also provides a central point where corporate data can be merged and integrated with data coming in from one or more Web sites into a common single source.

Chapter 11: ERP and the Data Warehouse

Overview

One of the most important advances in technology in the past 10 years has been Enterprise Resource Planning (ERP), the applications and transaction processing built for a company. The applications and transactions—built and maintained centrally by the vendor—cover the following corporate functions:

- Human resources
- Inventory management
- Financial management
- Supply chain management
- Logistics
- Manufacturing
- Distribution
- Customer service

ERP is led by such companies as SAP, PeopleSoft, JD Edwards, BAAN, Oracle Financials, and others. There are several reasons why organizations find ERP application software attractive. The primary reasons focus on the frustrations in handling existing applications. These include the following:

Aging. Many organizations have applications that were written years ago. These applications represent the business of a quarter century ago and were written so that they are unable to be changed. They are undocumented ande fragile. The people who wrote them have long departed.

Unintegrated. Existing applications each take a narrow view of the business. And no two applications have the same definitions, the same calculations, the same encoding structure, the same key structure as any other application. When a corporate perspective of data is needed, there is none.

Incomplete. Existing applications do not take into account such important events as year 2000 and the movement of currency to the Euro.

ERP software encompasses the transaction component of the business. At the transaction level the application governs the detailed interaction between the customer and the corporation. Payments are made, orders are entered, shipments are done, and products are manufactured.

ERP applications are at the heart of running any business. But with ERP, as with the non-ERP environment, transactions are only the start of the information process. There is still a need for a data warehouse.

ERP Applications Outside the Data Warehouse

Figure 11.1 shows the classical structuring of the data warehouse. This includes operational applications that feed data to a transformation process, the data warehouse, and DSS processes such as data marts, DSS applications, and exploration and data mining warehouses. The basic architecture of the data warehouse does not change in the face of ERP.

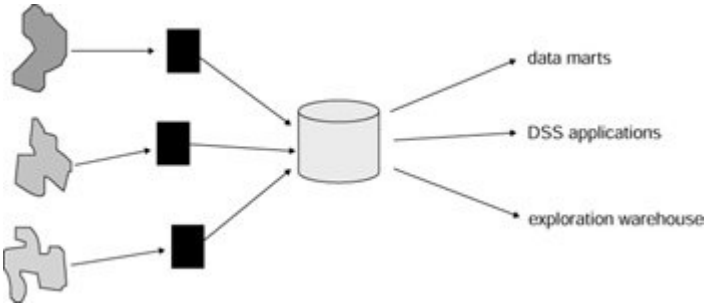

Figure 11.1: The data warehouse comes from operational applications and services other DSS components.

The ERP environment contains the application processing, where transactions occur. It then feeds the data warehouse, just like any other application. Figure 11.2 shows that ERP applications feed their data into a data warehouse.

The interface that takes data into the data warehouse is very similar to a non-ERP interface, with a few exceptions:

- The ERP interface has to deal with only the DBMS technologies that the ERP supports. The non-ERP interface must deal with all interfaces.
- It has a much tighter grip on the data and structures that constitute the application environment because the ERP vendor owns and controls the interface.
- The interface often has to go into the ERP environment and find the right data and "glue" it together in order to make it useful for the data warehouse environment.

Other than these differences, the ERP integration interface is the same as the interface that integrates data into a warehouse for non-ERP data. The data warehouse can be built in the ERP environment, as shown in Figure 11.2. Here data is pulled out of the ERP environment and into an operating environment completely unrelated to ERP. The data warehouse can be an Oracle, DB2, NT SQL Server, or other data warehouse. Once the data leaves the ERP environment, the ERP environment simply becomes another source of data.

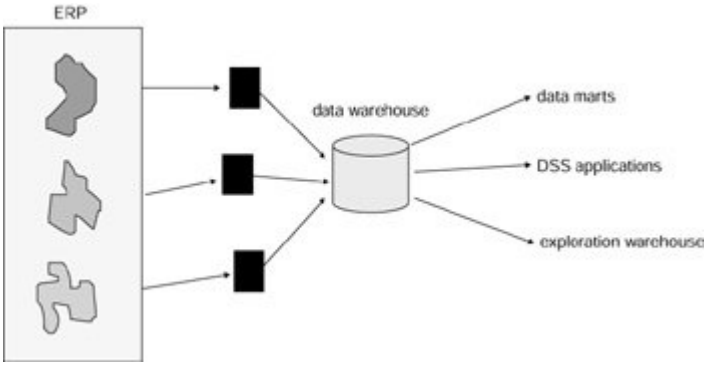

Figure 11.2: The ERP environment feeds the external data warehouse.

There are several advantages to pulling the data out of the ERP environment and into a different operating environment. One is that the data warehouse is free from any constraints that might be placed on it. Another is that the data warehouse database designer is free to design the data warehouse environment however he or she sees fit.

Building the Data Warehouse Inside the ERP Environment

The data warehouse can be built inside the confines of the ERP environment from ERP sources, as shown in Figure 11.3. Such a proposition might be SAP's BW or PeopleSoft's EPM. In this case, the ERP vendor provides the application and the data warehouse as well.

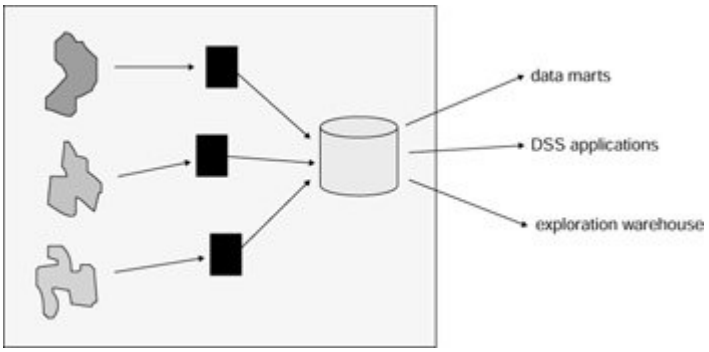

Figure 11.3: The data warehouse can be built in the ERP environment.

There are some distinct advantages to this approach. The primary advantage is that the ERP vendor provides the infrastructure. This saves the designer an enormous amount of work, and it saves the developer from having to deal with complexities of design and development. In short, the inclusion of the data warehouse with ERP applications greatly simplifies the life of the data warehouse designer. In addition, the long-term issue of maintenance is alleviated.

Feeding the Data Warehouse Through ERP and Non-ERP Systems

The inclusion of the data warehouse with the ERP applications is a popular choice because the customer has a solution. But there are other alternatives.

Figure 11.4 shows that a data warehouse is built outside the ERP environment and that non-ERP applications contribute data to the data warehouse.

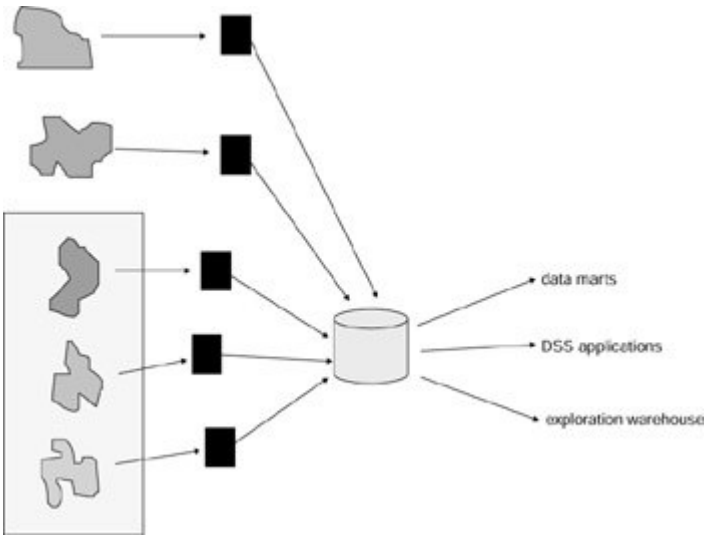

Figure 11.4: Both ERP and non-ERP operational systems support the non-ERP data warehouse.

Figure 11.4 is a very common scenario because of the issue of integrating non-ERP data with ERP data. Most organizations do not have a completely ERP environment. There are almost always non-ERP applications; where this occurs the two types of data must be integrated.

There is a second mitigating factor in the appeal of the structure seen in Figure 11.4. Often organizations build a data warehouse for their non-ERP data at the same time that the ERP environment is being established. By the time the ERP environment is complete (or at least functional), the non-ERP data warehouse is already established. When it comes time to put the ERP data in a data warehouse, it is a simple matter to take the ERP data out of the ERP environment and place it into the data warehouse.

But there is another alternative. Figure 11.5 shows that when the data warehouse is placed under the control of the ERP environment, external data can be placed into the data warehouse.

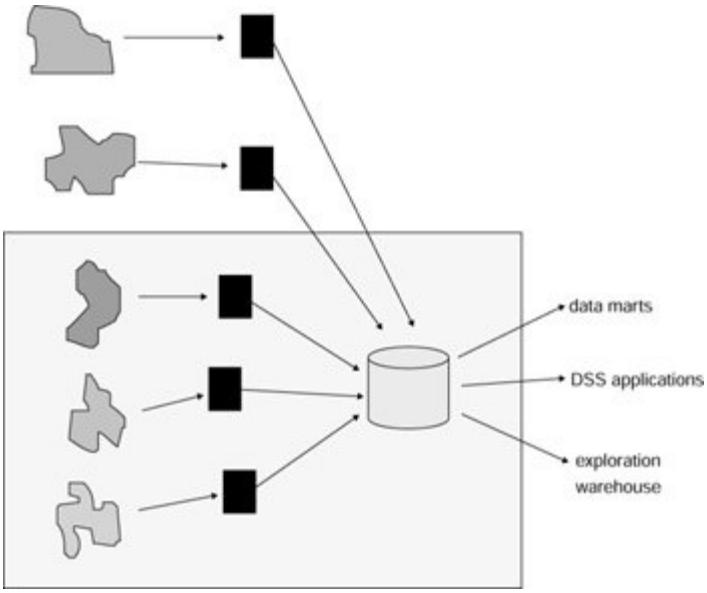

Figure 11.5: Both ERP and non-ERP operational systems support the ERP-based data warehouse.

Once the non-ERP data is placed in the control of the data warehouse, the data can take advantage of all the infrastructure built by the ERP vendor. This approach is most advantageous when the ERP environment is built before any data warehouse is built.

Putting all of the data in the ERP-controlled data warehouse does not mean that all DSS processing done from the data warehouse must be done within the confines of the ERP data warehouse. ERP vendors go out of their way to put a robust amount of analytical processing inside the ERP environment, and that is one of the great appeals of a ERP data warehouse. There is nothing, however, that says that other DSS processing cannot be done outside the ERP data warehouse environment. Figure 11.6 shows that DSS processing that relies on the data warehouse can be done either within or outside the ERP environment (or, for that matter, in both the ERP and non-ERP environments.)

There is yet another very common possibility for data warehousing in the ERP environment—a circumstance where there is both an ERP- and a non-

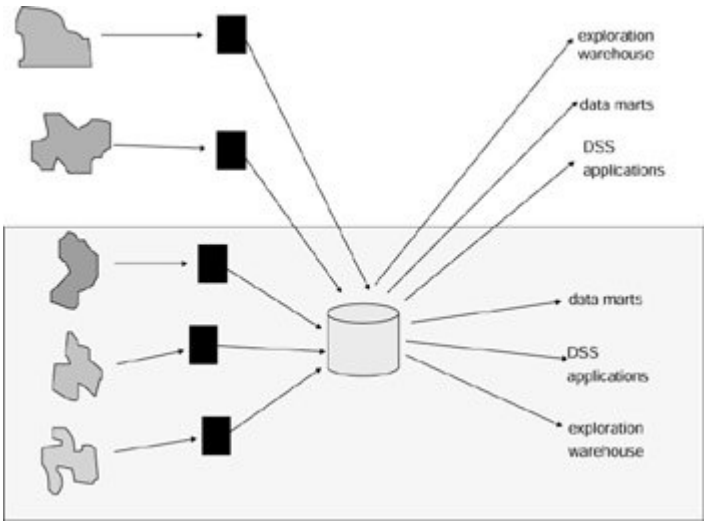

Figure 11.6: The data mart, DSS application, and exploration warehouse processing can be done inside or outside the ERP environment (or both) when the data warehouse is constructed inside the ERP environment.

ERP–based data warehouse, as shown in Figure 11.7. This possibility is quite legitimate when there is little or no business integration between the two environments. For example, the non-ERP data warehouse may represent a line of cosmetics, and the ERP-based data warehouse may represent the manufacture of motorcycles. Because there simply is no business relationship between cosmetics and motorcycles, it is quite acceptable to have separate data warehouses for the two environments.

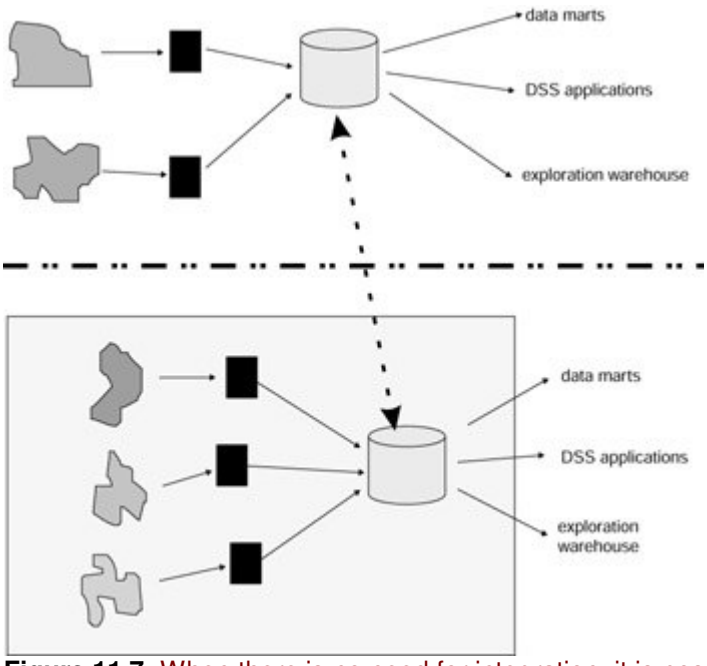

Figure 11.7: When there is no need for integration, it is possible to have multiple data warehouses.

If the two environments must share data, such as financial information, then it may be transferred from one warehouse to the other. For example, say that the cosmetics organization had quarterly profitability of $2,900,871 and the motorcycle manufacturer had

profitability for the quarter of $45,880,710. Because both organizations share a bottom line, there may be integration of data. But with regard to customers, business practices, promotions, campaigns, and so forth, there is absolutely no business integration. In such a case separate data warehouses make sense.

The ERP-Oriented Corporate Data Warehouse

Occasionally a specialized form of a global data warehouse—called the ERP-oriented corporate data warehouse—occurs. Figure 11.8 shows a corporate ERP data warehouse that collects data from the different geographic locales: the United States, Ethiopia, Switzerland, Japan, Australia, and Kuwait. The corporate data warehouse presents to management a truly global perspective of worldwide operations.

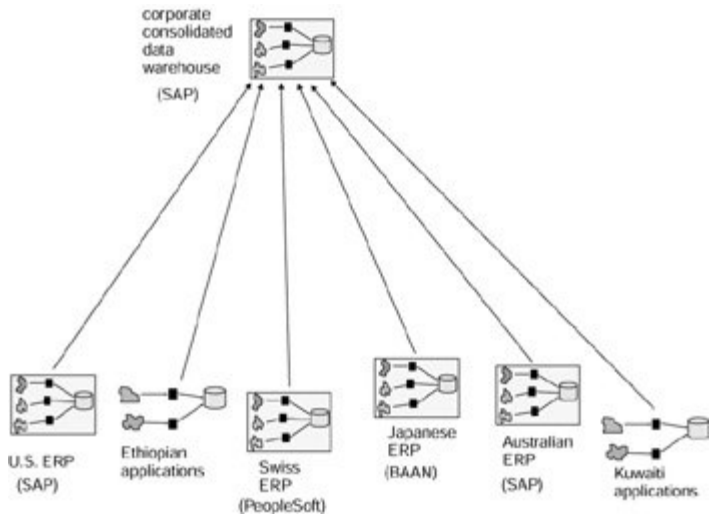

Figure 11.8: An ERP-oriented rendition of the global data warehouse.

The corporate data warehouse is in an ERP environment. So are most of the larger supporting local installations. But some of the local organizations have no ERP installation. How common is such a picture? Very common.

The first issue is getting data into the corporate ERP data warehouse. The data needs to be restructured and converted as it passes into the warehouse. How easy is the transition from the non-ERP installations to the corporate data warehouse? Not particularly. And how easy is the passage of data from a local ERP installation into the corporate ERP? Surprisingly difficult as well. The reason is that even though the local ERP installation may be in the same brand of software as the corporate ERP installation, there will nevertheless be many differences in the way the software has been implemented. Even if *all* of the local installations were in the same type of ERP, conversion to the headquarters installation would still be difficult.

Furthermore, one of the major challenges is keeping the headquarters data warehouse in sync. Perhaps one day the rebels take over in Peru and laws change. The headquarters data warehouse needs to be changed. The next day, a new discovery of oil is made in the Caribbean. More changes have to be made to the corporate data warehouse. Next month, a new electronic device is made that allows electricity generators to generate 10 times the power they once did with no change in equipment. The headquarters data warehouse needs to be changed again. Next month, there is an oil spill in Alaska, and the corporate data warehouse needs to be changed yet again.

In short, there is a never-ending series of events that require the corporate data to be constantly changed. Building the corporate infrastructure to withstand this eternal change is a significant challenge.

Summary

The ERP environment is an application-based environment found in many places. Even with an ERP installation there is a need for a data warehouse. The developer has several choices, of which each have advantages and disadvantages:

- Move ERP data to a non-ERP data warehouse.
- Move ERP data to an ERP-based data warehouse.
- Move ERP data to an ERP-based data warehouse and move non-ERP data to a non-ERP–based data warehous

Chapter 12: Data Warehouse Design Review Checklist

Overview

One of the most effective techniques for ensuring quality in the operational environment is the design review. Through design review, errors can be detected and resolved prior to coding. The cost benefit of identifying errors early in the development life cycle is enormous.

In the operational environment, design review is usually done on completion of the physical design of an application. The types of issues around which an operational design review centers are the following:

- Transaction performance
- Batch window adequacy
- System availability
- Capacity
- Project readiness
- User requirements satisfaction

Done properly in the operational environment, design review can save significant resources and greatly increase user satisfaction. Most importantly, when design review has been properly executed, major pieces of code do not have to be torn up and rewritten after the system has gone into production.

Design review is as applicable to the data warehouse environment as it is to the operational environment, with a few provisos.

One proviso is that systems are developed in the data warehouse environment in an iterative manner, where the requirements are discovered as a part of the development process. The classical operational environment is built under the well-defined system development life cycle (SDLC). Systems in the data warehouse environment are not built under the SDLC. Other differences between the development process in the operational environment and the data warehouse environment are the following:

- Development in the operational environment is done one application at a time. Systems for the data warehouse environment are built a subject area at a time.
- In the operational environment, there is a firm set of requirements that form the basis of operational design and development. In the data warehouse environment, there is seldom a firm understanding of processing requirements at the outset of DSS development.
- In the operational environment, transaction response time is a major and burning issue. In the data warehouse environment, transaction response time had better not be an issue.
- In the operational environment, the input from systems usually comes from sources external to the organization, most often from interaction with outside agencies. In the data warehouse environment, it usually comes from systems inside the organization where data is integrated from a wide variety of existing sources.
- In the operational environment, data is nearly all current valued (i.e., data is accurate as of the moment of use). In the data warehouse environment, data is time variant (i.e., data is relevant to some one moment in time).

There are, then, some substantial differences between the operational and data warehouse environments, and these differences show up in the way design review is conducted.

When to Do Design Review

Design review in the data warehouse environment is done as soon as a major subject area has been designed and is ready to be added to the data warehouse environment. It does not need to be done for every new database that goes up. Instead, as whole new major subject areas are added to the database, design review becomes an appropriate activity.

Who Should Be in the Design Review?

The attendees at the design review include anyone who has a stake in the development, operation, or use of the DSS subject area being reviewed.

Normally, this includes the following parties:

- The data administration (DA)
- The database administration (DBA)
- Programmers
- The DSS analysts
- End users other than the DSS analysts
- Operations
- Systems support
- Management

Of this group, by far the most important attendees are the end users and the DSS analysts.

One important benefit from having all the parties in the same room at the same time is the opportunity to short-circuit miscommunications. In an everyday environment where the end user talks to the liaison person who talks to the designer who talks to the programmer, there is ample opportunity for miscommunication and misinterpretation. When all the parties are gathered, direct conversations can occur that are beneficial to the health of the project being reviewed.

What Should the Agenda Be?

The subject for review for the data warehouse environment is any aspect of design, development, project management, or use that might prevent success. In short, any obstacle to success is relevant to the design review process. As a rule, the more controversial the subject, the more important that it be addressed during the review.

The questions that form the basis of the review process are addressed in the latter part of this chapter.

The Results

A data warehouse design review has three results:

- An appraisal to management of the issues, and recommendations as to further action
- A documentation of where the system is in the design, as of the moment of review
- An action item list that states specific objectives and activities that are a result of the review process

Administering the Review

The review is led by two people—a facilitator and a recorder. The facilitator is never the manager or the developer of the project being reviewed. If, by some chance, the facilitator is the project leader, the purpose of the review—from many perspectives—will have been defeated.

To conduct a successful review, the facilitator must be someone removed from the project for the following reasons:

As an outsider, the facilitator provides an external perspective—a fresh look—at the system. This fresh look often reveals important insights that someone close to the design and development of the system is not capable of providing.

As an outsider, a facilitator can offer criticism constructively. The criticism that comes from someone close to the development effort is usually taken personally and causes the design review to be reduced to a very base level.

A Typical Data Warehouse Design Review

1. 1. Who is missing in the review? Is any group missing that ought to be in attendance? Are the following groups represented?
    - DA
    - DBA
    - Programming
    - DSS analysts
    - End users
    - Operations
    - Systems programming
    - Auditing
    - Management

    Who is the official representative of each group?

    ISSUE: The proper attendance at the design review by the proper people is vital to the success of the review regardless of any other factors. Easily, the

    most important attendee is the DSS analyst or the end user. Management may or may not attend at their discretion.
2. 2. Have the end-user requirements been anticipated at all? If so, to what extent have they been anticipated? Does the end-user representative to the design review agree with the representation of requirements that has been done?

    ISSUE: In theory, the DSS environment can be built without interaction with the end user—with no anticipation of end-user requirements. If there will be a need to change the granularity of data in the data warehouse environment, or if EIS/artificial intelligence processing is to be built on top of the data warehouse, then some anticipation of requirements is a healthy exercise to go through. As a rule, even when the DSS requirements are anticipated, the level of participation of the end users is very low, and the end result is very sketchy. Furthermore, a large amount of time should not be allocated to the anticipation of end-user requirements.
3. 3. How much of the data warehouse has already been built in the data warehouse environment?
    - Which subjects?
    - What detail? What summarization?
    - How much data—in bytes? In rows? In tracks/cylinders?
    - How much processing?

- What is the growth pattern, independent of the project being reviewed?

ISSUE: The current status of the data warehouse environment has a great influence on the development project being reviewed. The very first development effort should be undertaken on a limited-scope, trial-and-error basis. There should be little critical processing or data in this phase. In addition, a certain amount of quick feedback and reiteration of development should be anticipated.

Later efforts of data warehouse development will have smaller margins for error.

4. 4. How many major subjects have been identified from the data model? How many are currently implemented? How many are fully implemented? How many are being implemented by the development project being reviewed? How many will be implemented in the foreseeable future?

   ISSUE: As a rule, the data warehouse environment is implemented one subject at a time. The first few subjects should be considered almost as experiments. Later subject implementation should reflect the lessons learned from earlier development efforts.

5. 5. Does any major DSS processing (i.e., data warehouse) exist outside the data warehouse environment? If so, what is the chance of conflict or overlap? What migration plan is there for DSS data and processing outside the data warehouse environment? Does the end user understand the migration that will have to occur? In what time frame will the migration be done?

   ISSUE: Under normal circumstances, it is a major mistake to have only part of the data warehouse in the data warehouse environment and other parts out of the data warehouse environment. Only under the most exceptional circumstances should a "split" scenario be allowed. (One of those circumstances is a distributed DSS environment.)

   If part of the data warehouse, in fact, does exist outside the data warehouse environment, there should be a plan to bring that part of the DSS world back into the data warehouse environment.

6. 6. Have the major subjects that have been identified been broken down into lower levels of detail?
   - Have the keys been identified?
   - Have the attributes been identified?
   - Have the keys and attributes been grouped together?
   - Have the relationships between groupings of data been identified?
   - Have the time variances of each group been identified?

   ISSUE: There needs to be a data model that serves as the intellectual heart of the data warehouse environment. The data model normally has three levels—a high-level model where entities and relationships are identified; a midlevel where keys, attributes, and relationships are identified; and a low level, where database design can be done. While not all of the data needs to be modeled down to the lowest level of detail in order for the DSS environment to begin to be built, at least the high-level model must be complete.

7. 7. Is the design discussed in question 6 periodically reviewed? (How often? Informally? Formally?) What changes occur as a result of the review? How is end-user feedback channeled to the developer?

   ISSUE: From time to time, the data model needs to be updated to reflect changing business needs of the organization. As a rule, these changes are incremental in nature. It is very unusual to have a revolutionary change. There needs to be an

assessment of the impact of these changes on both existing data warehouse data and planned data warehouse data.

8. 8. Has the operational system of record been identified?
    - Has the source for every attribute been identified?
    - Have the conditions under which one attribute or another will be the source been identified?
    - If there is no source for an attribute, have default values been identified?
    - Has a common measure of attribute values been identified for those data attributes in the data warehouse environment?
    - Has a common encoding structure been identified for those attributes in the data warehouse environment?
    - Has a common key structure in the data warehouse environment been identified? Where the system of record key does not meet the conditions for the DSS key structure, has a conversion path been identified?
    - If data comes from multiple sources, has the logic to determine the appropriate value been identified?
    - Has the technology that houses the system of record been identified?
    - Will any attribute have to be summarized on entering the data warehouse?
    - Will multiple attributes have to be aggregated on entering the data warehouse?
    - Will data have to be resequenced on passing into the data warehouse?

    ISSUE: After the data model has been built, the system of record is identified. The system of record normally resides in the operational environment. The system of record represents the best source of existing data in support of the data model. The issues of integration are very much a factor in defining the system of record.

9. 9. Has the frequency of extract processing—from the operational system of record to the data warehouse environment—been identified? How will the extract processing identify changes to the operational data from the last time an extract process was run?
    - By looking at time-stamped data?
    - By changing operational application code?
    - By looking at a log file? An audit file?
    - By looking at a delta file?
    - By rubbing “before” and “after” images together?

    ISSUE: The frequency of extract processing is an issue because of the resources required in refreshment, the complexity of refreshment processing, and the need to refresh data on a timely basis. The usefulness of data warehouse data is often related to how often the data warehouse data is refreshed.

    One of the most complex issues—from a technical perspective—is determining what data is to be scanned for extract processing. In some cases, the operational data that needs to pass from one environment to the next is straightforward. In other cases, it is not clear at all just what data should be examined as a candidate for populating the data warehouse environment.

10. 10. What volume of data will normally be contained in the DSS environment? If the volume of data is large,
    - Will multiple levels of granularity be specified?
    - Will data be compacted?
    - Will data be purged periodically?
    - Will data be moved to near-line storage? At what frequency?

ISSUE: In addition to the volumes of data processed by extraction, the designer needs to concern himself or herself with the volume of data actually in the data warehouse environment. The analysis of the volume of data in the data warehouse environment leads directly to the subject of the granularity of data in the data warehouse environment and the possibility of multiple levels of granularity.

11. 11. What data will be filtered out of the operational environment as extract processing is done to create the data warehouse environment?

    ISSUE: It is very unusual for all operational data to be passed to the DSS environment. Almost every operational environment contains data that is relevant only to the operational environment. This data should not be passed to the data warehouse environment.

12. 12. What software will be used to feed the data warehouse environment?
    - Has the software been thoroughly shaken out?
    - What bottlenecks are there or might there be?
    - Is the interface one-way or two-way?
    - What technical support will be required?
    - What volume of data will pass through the software?
    - What monitoring of the software will be required?
    - What alterations to the software will be periodically required?
    - What outage will the alterations entail?
    - How long will it take to install the software?
    - Who will be responsible for the software?
    - When will the software be ready for full-blown use?

    ISSUE: The data warehouse environment is capable of handling a large number of different types of software interfaces. The amount of break-in time and "infrastructure" time, however, should not be underestimated. The DSS architect must not assume that the linking of the data warehouse environment to other environments will necessarily be straightforward and easy.

13. 13. What software/interface will be required for the feeding of DSS departmental and individual processing out of the data warehouse environment?
    - Has the interface been thoroughly tested?
    - What bottlenecks might exist?
    - Is the interface one-way or two-way?
    - What technical support will be required?
    - What traffic of data across the interface is anticipated?
    - What monitoring of the interface will be required?
    - What alterations to the interface will there be?
    - What outage is anticipated as a result of alterations to the interface?
    - How long will it take to install the interface?
    - Who will be responsible for the interface?
    - When will the interface be ready for full-scale utilization?

14. 14. What physical organization of data will be used in the data warehouse environment? Can the data be directly accessed? Can it be sequentially accessed? Can indexes be easily and cheaply created?

    ISSUE: The designer needs to review the physical configuration of the data warehouse environment to ensure that adequate capacity will be available and that the data, once in the environment, will be able to be manipulated in a responsive manner.

15. 15. How easy will it be to add more storage to the data warehouse environment at a later point in time? How easy will it be to reorganize data within the data warehouse environment at a later point in time?

ISSUE: No data warehouse is static, and no data warehouse is fully specified at the initial moment of design. It is absolutely normal to make corrections in design throughout the life of the data warehouse environment. To construct a data warehouse environment either where midcourse corrections cannot be made or are awkward to make is to have a faulty design.

16. 16. What is the likelihood that data in the data warehouse environment will need to be restructured frequently (i.e., columns added, dropped, or enlarged, keys modified, etc.)? What effect will these activities of restructuring have on ongoing processing in the data warehouse?

    ISSUE: Given the volume of data found in the data warehouse environment, restructuring it is not a trivial issue. In addition, with archival data, restructuring after a certain moment in time often becomes a logical impossibility.

17. 17. What are the expected levels of performance in the data warehouse environment? Has a DSS service level agreement been drawn up either formally or informally?

    ISSUE: Unless a DSS service-level agreement has been formally drawn up, it is impossible to measure whether performance objectives are being met. The DSS service level agreement should cover both DSS performance levels and downtime. Typical DSS service level agreements state such things as the following:
    - Average performance during peak hours per units of data
    - Average performance during off-peak hours per units of data
    - Worst performance levels during peak hours per units of data
    - Worst performance during off-peak hours per units of data
    - System availability standards

    One of the difficulties of the DSS environment is measuring performance. Unlike the operational environment where performance can be measured in absolute terms, DSS processing needs to be measured in relation to the following:
    - How much processing the individual request is for
    - How much processing is going on concurrently
    - How many users are on the system at the moment of execution

18. 18. What are the expected levels of availability? Has an availability agreement been drawn up for the data warehouse environment, either formally or informally?

    ISSUE: (See issue for question 17.)

19. 19. How will the data in the data warehouse environment be indexed or accessed?
    - Will any table have more than four indexes?
    - Will any table be hashed?
    - Will any table have only the primary key indexed?
    - What overhead will be required to maintain the index?
    - What overhead will be required to load the index initially?
    - How often will the index be used?
    - Can/should the index be altered to serve a wider use?

    ISSUE: Data in the data warehouse environment needs to be able to be accessed efficiently and in a flexible manner. Unfortunately, the heuristic

    nature of data warehouse processing is such that the need for indexes is unpredictable. The result is that the accessing of data in the data warehouse environment must not be taken for granted. As a rule, a multitiered approach to managing the access of data warehouse data is optimal:
    - The hashed/primary key should satisfy most accesses.
    - Secondary indexes should satisfy other popular access patterns.
    - Temporary indexes should satisfy the occasional access.

- Extraction and subsequent indexing of a subset of data warehouse data should satisfy infrequent or once-in-a-lifetime accesses of data.

In any case, data in the data warehouse environment should not be stored in partitions so large that they cannot be indexed freely.

20. 20. What volumes of processing in the data warehouse environment are to be expected? What about peak periods? What will the profile of the average day look like? The peak rate?

ISSUE: Not only should the volume of data in the data warehouse environment be anticipated, but the volume of processing should be anticipated as well.

21. 21. What level of granularity of data in the data warehouse environment will there be?
    - A high level?
    - A low level?
    - Multiple levels?
    - Will rolling summarization be done?
    - Will there be a level of true archival data?
    - Will there be a living sample level of data?

ISSUE: Clearly, the most important design issue in the data warehouse environment is that of granularity of data and the possibility of multiple levels of granularity. In a word, if the granularity of the data warehouse environment is done properly, then all other issues become straightforward; if the granularity of data in the data warehouse environment is not designed properly, then all other design issues become complex and burdensome.

22. 22. What purge criteria for data in the data warehouse environment will there be? Will data be truly purged, or will it be compacted and archived elsewhere? What legal requirements are there? What audit requirements are there?

ISSUE: Even though data in the DSS environment is archival and of necessity has a low probability of access, it nevertheless has some probability of access (otherwise it should not be stored). When the probability of access reaches zero (or approaches zero), the data needs to be purged. Given that volume of data is one of the most burning issues in the data warehouse environment, purging data that is no longer useful is one of the more important aspects of the data warehouse environment.

23. 23. What total processing capacity requirements are there:
    - For initial implementation?
    - For the data warehouse environment at maturity?

ISSUE: Granted that capacity requirements cannot be planned down to the last bit, it is worthwhile to at least estimate how much capacity will be required, just in case there is a mismatch between needs and what will be available.

24. 24. What relationships between major subject areas will be recognized in the data warehouse environment? Will their implementation do the following:
    - Cause foreign keys to be kept up-to-date?
    - Make use of artifacts?

What overhead is required in the building and maintenance of the relationship in the data warehouse environment?

ISSUE: One of the most important design decisions the data warehouse designer makes is that of how to implement relationships between data in the data warehouse environment. Data relationships are almost never implemented the same way in the data warehouse as they are in the operational environment.

25. 25. Do the data structures internal to the data warehouse environment make use of the following:
    - Arrays of data?
    - Selective redundancy of data?
    - Merging of tables of data?
    - Creation of commonly used units of derived data?

    ISSUE: Even though operational performance is not an issue in the data warehouse environment, performance is nevertheless an issue. The designer needs to consider the design techniques listed previously when they can reduce the total amount of I/O consumed. The techniques listed previously are classical physical denormalization techniques. Because data is not updated in the data warehouse environment, there are very few restrictions on what can and can't be done.

    The factors that determine when one or the other design technique can be used include the following:
    - The predictability of occurrences of data
    - The predictability of the pattern of access of data
    - The need to gather artifacts of data
26. 26. How long will a recovery take? Is computer operations prepared to execute a full data warehouse database recovery? A partial recovery? Will operations periodically practice recovery so that it will be prepared in the event of a need for recovery? What level of preparedness is exhibited by the following:
    - Systems support?
    - Applications programming?
    - The DBA?
    - The DA?

    For each type of problem that can arise, is it clear whose responsibility the problem is?

    ISSUE: As in operational systems, the designer must be prepared for the outages that occur during recovery. The frequency of recovery, the length of time required to bring the system back up, and the domino effect that can occur during an outage must all be considered.

    Have instructions been prepared, tested, and written? Have these instructions been kept up-to-date?
27. 27. What level of preparation is there for reorganization/restructuring of:
    - Operations?
    - Systems support?
    - Applications programming?
    - The DBA?
    - The DA?

    Have written instructions and procedures been set and tested? Are they up-to-date? Will they be kept up-to-date?

    ISSUE: (See issues for question 26.)
28. 28. What level of preparation is there for the loading of a database table by:
    - Operations?
    - Systems support?
    - Applications programming?
    - The DBA?
    - The DA?

Have written instructions and procedures been made and tested? Are they up-to-date? Will they be kept up-to-date?

ISSUE: The time and resources for loading can be considerable. This estimate needs to be made carefully and early in the development life cycle.

29. 29. What level of preparation is there for the loading of a database index by:
    - Operations?
    - Systems support?
    - Applications programming?
    - The DBA?
    - The DA?

    ISSUE: (See issue for question 28.)
30. 30. If there is ever a controversy as to the accuracy of a piece of data in the data warehouse environment, how will the conflict be resolved? Has ownership (or at least source identification) been done for each unit of data in the data warehouse environment? Will ownership be able to be established if the need arises? Who will address the issues of ownership? Who will be the final authority as to the issues of ownership?

    ISSUE: Ownership or stewardship of data is an essential component of success in the data warehouse environment. It is inevitable that at some moment in time the contents of a database will come into question. The designer needs to plan in advance for this eventuality.
31. 31. How will corrections to data be made once data is placed in the data warehouse environment? How frequently will corrections be made? Will corrections be monitored? If there is a pattern of regularly occurring changes, how will corrections at the source (i.e., operational) level be made?

    ISSUE: On an infrequent, nonscheduled basis, there may need to be changes made to the data warehouse environment. If there appears to be a pattern to these changes, then the DSS analyst needs to investigate what is wrong in the operational system.
32. 32. Will public summary data be stored separately from normal primitive DSS data? How much public summary data will there be? Will the algorithm required to create public summary data be stored?

    ISSUE: Even though the data warehouse environment contains primitive data, it is normal for there to be public summary data in the data warehouse environment as well. The designer needs to have prepared a logical place for this data to reside.
33. 33. What security requirements will there be for the databases in the data warehouse environment? How will security be enforced?

    ISSUE: The access of data becomes an issue, especially as the detailed data becomes summarized or aggregated, where trends become apparent. The designer needs to anticipate the security requirements and prepare the data warehouse environment for them.
34. 34. What audit requirements are there? How will audit requirements be met?

    ISSUE: As a rule, system audit can be done at the data warehouse level, but this is almost always a mistake. Instead, detailed record audits are best done at the system-of-record level.
35. 35. Will compaction of data be used? Has the overhead of compacting/decompacting data been considered? What is the overhead? What are the savings in terms of DASD for compacting/decompacting data?

ISSUE: On one hand, compaction or encoding of data can save significant amounts of space. On the other hand, both compacting and encoding data require CPU cycles as data is decompacted or decoded on access. The designer needs to make a thorough investigation of these issues and a deliberate trade-off in the design.

36. 36. Will encoding of data be done? Has the overhead of encoding/decoding been considered? What, in fact, is the overhead?

ISSUE: (See issue for question 35.)

37. 37. Will meta data be stored for the data warehouse environment?

ISSUE: Meta data needs to be stored with any archival data as a matter of policy. There is nothing more frustrating than an analyst trying to solve a problem using archival data when he or she does not know the meaning of the contents of a field being analyzed. This frustration can be alleviated by storing the semantics of data with the data as it is archived. Over time, it is absolutely normal for the contents and structure of data in the data warehouse environment to change. Keeping track of the changing definition of data is something the designer should make sure is done.

38. 38. Will reference tables be stored in the data warehouse environment?

ISSUE: (See issue for question 37.)

39. 39. What catalog/dictionary will be maintained for the data warehouse environment? Who will maintain it? How will it be kept up-to-date? To whom will it be made available?

ISSUE: Not only is keeping track of the definition of data over time an issue, but keeping track of data currently in the data warehouse is important as well.

40. 40. Will update (as opposed to loading and access of data) be allowed in the data warehouse environment? (Why? How much? Under what circumstances? On an exception-only basis?)

ISSUE: If any updating is allowed on a regular basis in the data warehouse environment, the designer should ask why. The only update that should occur should be on an exception basis and for only small amounts of data. Any exception to this severely compromises the efficacy of the data warehouse environment.

When updates are done (if, in fact, they are done at all), they should be run in a private window when no other processing is done and when there is slack time on the processor.

41. 41. What time lag will there be in getting data from the operational to the data warehouse environment? Will the time lag ever be less than 24 hours? If so, why and under what conditions? Will the passage of data from the operational to the data warehouse environment be a “push” or a “pull” process?

ISSUE: As a matter of policy, any time lag less than 24 hours should be questioned. As a rule, if a time lag of less than 24 hours is required, it is a sign that the developer is building operational requirements into the data warehouse. The flow of data through the data warehouse environment should always be a pull process, where data is pulled into the warehouse environment when it is needed, rather than being pushed into the warehouse environment when it is available.

42. 42. What logging of data warehouse activity will be done? Who will have access to the logs?

ISSUE: Most DSS processing does not require logging. If an extensive amount of logging is required, it is usually a sign of lack of understanding of what type of processing is occurring in the data warehouse environment.

43. 43. Will any data other than public summary data flow to the data warehouse environment from the departmental or individual level? If so, describe it.

    ISSUE: Only on rare occasions should public summary data come from sources other than departmental or individual levels of processing. If much public summary data is coming from other sources, the analyst should ask why.
44. 44. What external data (i.e., data other than that generated by a company's internal sources and systems) will enter the data warehouse environment? Will it be specially marked? Will its source be stored with the data? How frequently will the external data enter the system? How much of it will enter? Will an unstructured format be required? What happens if the external data is found to be inaccurate?

    ISSUE: Even though there are legitimate sources of data other than a company's operational systems, if much data is entering externally, the analyst should ask why. Inevitably, there is much less flexibility with the content and regularity of availability of external data, although external data represents an important resource that should not be ignored.
45. 45. What facilities will exist that will help the departmental and the individual user to locate data in the data warehouse environment?

    ISSUE: One of the primary features of the data warehouse is ease of accessibility of data. And the first step in the accessibility of data is the initial location of the data.
46. 46. Will there be an attempt to mix operational and DSS processing on the same machine at the same time? (Why? How much processing? How much data?)

    ISSUE: For a multitude of reasons, it makes little sense to mix operational and DSS processing on the same machine at the same time. Only where there are small amounts of data and small amounts of processing should there be a mixture. But these are not the conditions under which the data warehouse environment is most cost-effective. (See my previous book *Data Architecture: The Information Paradigm* [QED/Wiley, 1992] for an in-depth discussion of this issue.)
47. 47. How much data will flow back to the operational level from the data warehouse level? At what rate? At what volume? Under what response time constraints? Will the flowback be summarized data or individual units of data?

    ISSUE: As a rule, data flows from the operational to the warehouse level to the departmental to the individual levels of processing. There are some notable exceptions. As long as not too much data "backflows," and as long as the backflow is done in a disciplined fashion, there usually is no problem. If there is a lot of data engaged in backflow, then a red flag should be raised.
48. 48. How much repetitive processing will occur against the data warehouse environment? Will precalculation and storage of derived data save processing time?

    ISSUE: It is absolutely normal for the data warehouse environment to have some amount of repetitive processing done against it. If only repetitive processing is done, however, or if no repetitive processing is planned, the designer should question why.
49. 49. How will major subjects be partitioned? (By year? By geography? By functional unit? By product line?) Just how finely does the partitioning of the data break the data up?

    ISSUE: Given the volume of data that is inherent to the data warehouse environment and the unpredictable usage of the data, it is mandatory that data warehouse data be partitioned into physically small units that can be managed independently. The design issue is not whether partitioning is to be done. Instead, the design issue is how partitioning is to be accomplished. In general, partitioning is done at the application level rather than the system level.

The partitioning strategy should be reviewed with the following in mind:

- Current volume of data
- Future volume of data
- Current usage of data
- Future usage of data
- Partitioning of other data in the warehouse
- Use of other data
- Volatility of the structure of data

50. 50. Will sparse indexes be created? Would they be useful?

    ISSUE: Sparse indexes created in the right place can save huge amounts of processing. By the same token, sparse indexes require a fair amount of overhead in their creation and maintenance. The designer of the data warehouse environment should consider their use.
51. 51. What temporary indexes will be created? How long will they be kept? How large will they be?

    ISSUE: (See the issue for question 50, except as it applies to temporary indexes.)
52. 52. What documentation will there be at the departmental and individual levels? What documentation will there be of the interfaces between the data warehouse environment and the departmental environment? Between the departmental and the individual environment? Between the data warehouse environment and the individual environment?

    ISSUE: Given the free-form nature of processing in the departmental and the individual environments, it is unlikely that there will be much in the way of available documentation. A documentation of the relationships between the environments is important for the reconcilability of data.
53. 53. Will the user be charged for departmental processing? For individual processing? Who will be charged for data warehouse processing?

    ISSUE: It is important that users have their own budgets and be charged for resources used. The instant that processing becomes “free,” it is predictable that there will be massive misuse of resources. A chargeback system instills a sense of responsibility in the use of resources.
54. 54. If the data warehouse environment is to be distributed, have the common parts of the warehouse been identified? How are they to be managed?

    ISSUE: In a distributed data warehouse environment, some of the data will necessarily be tightly controlled. The data needs to be identified up front by the designer and meta data controls put in place.
55. 55. What monitoring of the data warehouse will there be? At the table level? At the row level? At the column level?

    ISSUE: The use of data in the warehouse needs to be monitored to determine the dormancy rate. Monitoring must occur at the table level, the row level, and the column level. In addition, monitoring of transaction needs to occur as well.
56. 56. Will class IV ODS be supported? How much performance impact will there be on the data warehouse to support class IV ODS processing?

    ISSUE: Class IV ODS is fed from the data warehouse. The data needed to create the profile in the class IV ODS is found in the data warehouse.
57. 57. What testing facility will there be for the data warehouse?

    ISSUE: Testing in the data warehouse is not the same level of importance as in the operational transaction environment. But occasionally there is a need for testing,

especially when new types of data are being loaded and when there are large volumes of data.

58. 58. What DSS applications will be fed from the data warehouse? How much volume of data will be fed?

    ISSUE: DSS applications, just like data marts, are fed from the data warehouse. There are the issues of when the data warehouse will be examined, how often it will be examined, and what performance impact there will be because for the analysis.

59. 59. Will an exploration warehouse and/or a data mining warehouse be fed from the data warehouse? If not, will exploration processing be done directly in the data warehouse? If so, what resources will be required to feed the exploration/data mining warehouse?

    ISSUE: The creation of an exploration warehouse and/or a data mining data warehouse can greatly alleviate the resource burden on the data warehouse. An exploration warehouse is needed when the frequency of exploration is such that statistical analysis starts to have an impact on data warehouse resources.

    The issues here are the frequency of update and the volume of data that needs to be updated. In addition, the need for an incremental update of the data warehouse occasionally arises.

60. 60. What resources are required for loading data into the data warehouse on an ongoing basis? Will the load be so large that it cannot fit into the window of opportunity? Will the load have to be parallelized?

    ISSUE: Occasionally there is so much data that needs to be loaded into the data warehouse that the window for loading is not large enough. When the load is too large there are several options:

    - Creating a staging area where much preprocessing of the data to be loaded can be done independently
    - Parallelizing the load stream so that the elapsed time required for loading is shrunk to the point that the load can be done with normal processing
    - Editing or summarizing the data to be loaded so that the actual load is smaller

61. 61. To what extent has the midlevel model of the subject areas been created? Is there a relationship between the different midlevel models?

    ISSUE: Each major subject area has its own midlevel data model. As a rule the midlevel data models are created only as the iteration of development needs to have them created. In addition, the midlevel data models are related in the same way that the major subject areas are related.

62. 62. Is the level of granularity of the data warehouse sufficiently low enough in order to service all the different architectural components that will be fed from the data warehouse?

    ISSUE: The data warehouse feeds many different architectural components. The level of granularity of the data warehouse must be sufficiently low to feed the lowest level of data needed anywhere in the corporate information factory. This is why it is said that the data in the data warehouse is at the lowest common denominator.

63. 63. If the data warehouse will be used to store ebusiness and clickstream data, to what extent does the Granularity Manager filter the data?

    ISSUE: The Web-based environment generates a huge amount of data. The data that is generated is at much too low a level of granularity. In order to summarize and aggregate the data before entering the data warehouse, the data is passed through a

Granularity Manager. The Granularity Manager greatly reduces the volume of data that finds its way into the data warehouse.

64. 64. What dividing line is used to determine what data is to be placed on disk storage and what data is to be placed on alternate storage?

    ISSUE: The general approach that most organizations take in the placement of data on disk storage and data on alternate storage is to place the most current data on disk storage and to place older data on alternate storage. Typically, disk storage may hold two years' worth of data, and alternate storage may hold all data that is older than two years.
65. 65. How will movement of data to and from disk storage and alternate storage be managed?

    ISSUE: Most organizations have software that manages the traffic to and from alternate storage. The software is commonly known as a cross-media storage manager.
66. 66. If the data warehouse is a global data warehouse, what data will be stored locally and what data will be stored globally?

    ISSUE: When a data warehouse is global, some data is stored centrally and other data is stored locally. The dividing line is determined by the use of the data.
67. 67. For a global data warehouse, is there assurance that data can be transported across international boundaries?

    ISSUE: Some countries have laws that do not allow data to pass beyond their boundaries. The data warehouse that is global must ensure that it is not in violation of international laws.
68. 68. For ERP environments, has it been determined where the data warehouse will be located—inside the ERP software or outside the ERP environment?

    ISSUE: Many factors determine where the data warehouse should be placed:
    - Does the ERP vendor support data warehouse?
    - Can non-ERP data be placed inside the data warehouse?
    - What analytical software can be used on the data warehouse if the data warehouse is placed inside the ERP environment?
    - If the data warehouse is placed inside the ERP environment, what DBMS can be used?
69. 69. Can alternate storage be processed independently?

    ISSUE: Older data is placed in alternate storage. It is often quite useful to be able to process the data found in alternate storage independently of any consideration of data placed on disk storage.
70. 70. s the development methodology that is being used for development a spiral development approach or a classical waterfall approach?

    ISSUE: The spiral development approach is always the correct development approach for the data warehouse environment. The waterfall SDLC approach is never the appropriate approach.
71. 71. Will an ETL tool be used for moving data from the operational environment to the data warehouse environment, or will the transformation be done manually?

    ISSUE: In almost every case, using a tool of automation to transform data into the data warehouse environment makes sense. Only where there is a very small amount of data to be loaded into the data warehouse environment should the loading of the data warehouse be done manually.

Summary

Design review is an important quality assurance practice that can greatly increase the satisfaction of the user and reduce development and maintenance costs. Thoroughly reviewing the many aspects of a warehouse environment prior to building the warehouse is a sound practice.

The review should focus on both detailed design and architecture.

Appendix

DEVELOPING OPERATIONAL SYSTEMS—METH 1

M1—Initial Project Activities

PRECEDING ACTIVITY: Decision to build an operational system.

FOLLOWING ACTIVITY: Preparing to use existing code/data.

TIME ESTIMATE: Indeterminate, depending on size of project.

NORMALLY EXECUTED ONCE OR MULTIPLE TIMES: Once.

SPECIAL CONSIDERATIONS: Because of the ambiguity of this step, it tends to drag out interminably. As long as 90 percent (or even less) of the system is defined here, the system development should continue into the next phase.

DELIVERABLE: Raw system requirements.

Interviews. The output of interviews is the "softcore" description of what the system is to do, usually reflecting the opinion of middle management. The format of the output is very free-form. As a rule, the territory covered by interviews is not comprehensive.

Data gathering. The output from this activity may come from many sources. In general, requirements-usually detailed-that are not caught elsewhere are gathered here. This is a free-form, catchall, requirements-gathering activity, the results of which fill in the gap for other requirements-gathering activities.

JAD (Joint Application Design) session output. The output from these activities is the group "brainstorm" synopsis. Some of the benefits of requirements formulation in a JAD session are the spontaneity and flow of ideas, and the critical mass that occurs by having different people in the same room focusing on a common objective. The output of one or more JAD sessions is a formalized set of requirements that collectively represent the end users' needs.

Strategic business plan analysis. If the company has a strategic business plan, it makes sense to reflect on how the plan relates to the requirements of the system being designed. The influence of the strategic business plan can manifest itself in many ways-in setting growth figures, in identifying new lines of business, in describing organizational changes, and so forth. All of these factors, and more, shape the requirements of the system being built.

Existing systems shape requirements for a new system profoundly. If related, existing systems have been built, at the very least the interface between the new set of requirements and existing systems must be identified.

Conversion, replacement, parallel processing, and so forth are all likely topics. The output of this activity is a description of the impact and influence of existing systems on the requirements for the system being developed.

PARAMETERS OF SUCCESS: When done properly, there is a reduction in the ambiguity of the system, the scope of the development effort is reasonably set, and the components of

the system are well organized. The political as well as the technical components of the system should be captured and defined.

M2—Using Existing Code/Data

PRECEDING ACTIVITY: System definition.

FOLLOWING ACTIVITY: Sizing, phasing.

TIME ESTIMATE: Done very quickly, usually in no more than a week in even the largest of designs.

NORMALLY EXECUTED ONCE OR MULTIPLE TIMES: Once.

SPECIAL CONSIDERATIONS: This step is one of the best ways to ensure code reusability and data reusability. This step is crucial to the integration of the environment.

In an architected environment, it is incumbent on every project to do the following:

- Use as much existing code/data as possible.
- Prepare for future projects that will use code and data to be developed in the current project. The output from this step is an identification of existing code/data that can be reused and the steps that need to be taken for future processing.

If existing code/data is to be modified, the modifications are identified as a regular part of the system development requirements. If existing code/data is to be deleted, the deletion becomes a part of the specifications. If conversion of code/data are to be done, the conversion becomes a component of the development effort.

PARAMETERS OF SUCCESS: To identify code/data that already exists that can be built on; to identify what needs to be built to prepare for future efforts.

M3—Sizing, Phasing

PRECEDING ACTIVITY: Using existing code/data.

FOLLOWING ACTIVITY: Requirements formalization; capacity analysis.

TIME ESTIMATE: This step goes rapidly, usually in a day or two, even for the largest of designs.

NORMALLY EXECUTED ONCE OR MULTIPLE TIMES: Once, then revisited for each continuing phase of development.

DELIVERABLE: Identification of phases of development.

After the general requirements are gathered, the next step is to size them and divide development up into phases. If the system to be developed is large, it makes sense to break it into development phases. In doing so, development is parceled out in small, manageable units. Of course, the different development phases must be organized into a meaningful sequence, so that the second phase builds on the first, the third phase builds on the first and second, and so on.

The output from this step is the breakup of general requirements into doable, manageable phases, if the requirements are large enough to require a breakup at all.

PARAMETERS OF SUCCESS: To continue the development process in increments that are both economic and doable (and within the political context of the organization as well).

M4—Requirements Formalization

PRECEDING ACTIVITY: Sizing, phasing.

FOLLOWING ACTIVITY: ERD specification; functional decomposition.

TIME ESTIMATE: Indeterminate, depending on size of system, how well the scope of the system has been defined, and how ambiguous the design is up to this point.

NORMALLY EXECUTED ONCE OR MULTIPLE TIMES: Once per phase of development.

DELIVERABLE: Formal requirements specification.

Once the requirements have been gathered, sized, and phased (if necessary), the next step is to formalize them. In this step, the developer ensures the following:

- The requirements that have been gathered are complete, as far as it is reasonably possible to gather them.
- The requirements are organized.
- The requirements are readable, comprehensible, and at a low enough level of detail to be effective.
- The requirements are not in conflict with each other.
- The requirements do not overlap.
- Operational and DSS requirements have been separated.
- The output from this step is a formal requirements definition that is ready to go to detailed design.

PARAMETERS OF SUCCESS: A succinct, organized, readable, doable, quantified, complete, usable set of requirements that is also a document for development.

CA—Capacity Analysis

PRECEDING ACTIVITY: Sizing, phasing.

FOLLOWING ACTIVITY: ERD specification; functional decomposition.

TIME ESTIMATE: Depends on the size of the system being built, but with estimating tools and a focused planner, two or three weeks is a reasonable estimate for a reasonably sized system.

NORMALLY EXECUTED ONCE OR MULTIPLE TIMES: Once per phase of development.

SPECIAL CONSIDERATIONS: The capacity planning function has a history of confusing issues and including extraneous factors that do not merit special attention. It is important to keep the capacity planning portion of the development process focused and to the point. Otherwise, the exercise can become a roadblock to progress.

The gross amounts of resources to be consumed by the project being analyzed need to be estimated in this early phase of development. In particular, the following needs to be considered:

- DASD consumption
- Software requirements (including system software, utilities, special custom code, network software, interface software)

- CPU consumption
- I/O utilization
- Main memory requirements
- Network/channel utilization

Not only are raw requirements analyzed, but the arrival rate of transactions, peak-period processing, patterns of processing, response time requirements, availability requirements, and mean time to failure requirements are factored in as well.

In addition, if any hardware/software must be ordered, the lead time and the “burn in” time must be accounted for to ensure that proper resources will be in place in time for the application being reviewed.

The output of this phase of development is the assurance that the resources needed to be in place are, in fact, in place.

PARAMETERS OF SUCCESS: No surprises when it comes to resources being in place when needed, the lead time needed to acquire resources, and the amount of resources needed.

PREQ1—Technical Environment Definition

PRECEDING ACTIVITY: Establishment of the information processing environment.

FOLLOWING ACTIVITY: Sizing, phasing.

TIME ESTIMATE: NA

NORMALLY EXECUTED ONCE OR MULTIPLE TIMES: NA

SPECIAL CONSIDERATIONS: On occasion, it is necessary to build an application system from scratch, including defining the technical environment. If this is the case, the considerations of technical definition are outside the boundaries of the data-driven development methodology.

In order to proceed, it is necessary that the technical environment be defined. If the technical environment is not defined at this point, the result will be much “thrashing.” Simply stated, detailed design cannot be meaningfully done until the technical environment is defined.

Certain elements of design beyond this point depend on one or the other of the technical cornerstones previously identified. At the least, the following should be established:
- The hardware platform(s) that will be used
- The operating system(s) that will be used
- The DBMS(s) that will be used
- The network software to be used
- The language(s) to be used for development

In addition to whatever hardware, software, and networking will be used, it is helpful to establish the following as well:
- Node residency definition (for network systems)
- Management of the system of record

PARAMETERS OF SUCCESS: A firm, workable, technical definition that will meet the needs of the system being developed.

D1—ERD (Entity Relationship Diagram)

PRECEDING ACTIVITY: Requirements formalization.

FOLLOWING ACTIVITY: Data item set specification.

TIME ESTIMATE: For even the largest of systems, two weeks suffice if the designers know what they are doing. If they don't, the amount of time required here is indeterminate.

NORMALLY EXECUTED ONCE OR MULTIPLE TIMES: Once.

DELIVERABLE: Identification of major subject areas.

From the general set of formal requirements comes the need to identify the major subjects that will make up the system and the relationship of those major subjects. As a rule, the major subject is at the highest level of abstraction.

Typical major subjects are CUSTOMER, PRODUCT, TRANSACTION, and so forth. The relationships of the major subjects are identified, as well as the cardinality of the relationship.

The output of this step is the identification of the major subjects that will make up the system, as well as their relationships to each other.

PARAMETERS OF SUCCESS: All major subjects are identified so that there are no conflicts in domain; they are identified at the highest level of abstraction.

One major parameter of success is that only primitive data be modeled. Another parameter of success is that the scope of the model be defined prior to starting the ERD modeling.

D2—DIS (Data Item Sets)

PRECEDING ACTIVITY: ERD definition.

FOLLOWING ACTIVITY: Performance analysis; data store definition.

TIME ESTIMATE: As long as one month per subject area.

NORMALLY EXECUTED ONCE OR MULTIPLE TIMES: Once for each subject area.

Each subject is further broken down-in terms of level of detail-into a dis (data item set). The dis contains attributes of data, the grouping of attributes, and keys. In addition, "type of" data is identified. Other structures of data here include connectors-representations of relationships-and secondary groupings of data. The output from this step is the fleshing out of the subject areas identified in D1.

PARAMETERS OF SUCCESS: All types of the major subject are identified; all connectors are correctly identified; all relationships are identified by a connector; all attributes are identified; all attributes are grouped with other attributes that share the same relationship to the key of the grouping of data; all multiply occurring groups of attributes are separated from singularly occurring groups of attributes; all recursive relationships are designed in the most general case necessary. Only primitive data is found here. Derived data is identified, stored, and managed elsewhere.

D3—Performance Analysis

PRECEDING ACTIVITY: Data item set development.

FOLLOWING ACTIVITY: Physical database design.

TIME ESTIMATE: One week per subject area, unless the subject area is huge.

NORMALLY EXECUTED ONCE OR MULTIPLE TIMES: Once per subject area.

SPECIAL CONSIDERATIONS: This step does not need to be done in the case of small amounts of data and/or small amounts of processing.

This step is performed if the volume of data, the volume of processing, the traffic over the network, the growth of data and processing, or the peak period of processing will produce significant amounts of activity. If none of those factors will occur, this step is not done.

In this step, the issue of physical denormalization of data is addressed. Specifically, the design practices of merging tables, selective introduction of redundancy, creating popular pools of derived data, creating arrays of data, and further separation of data where there is a wide disparity in the probability of its access are considered.

If this activity is done at all, the output reflects a much more streamlined design, with little or no loss of the benefits of normalization of data.

PARAMETERS OF SUCCESS: A design that will be efficient to access and update, in terms of both data and programs that access and update data. Done properly, this step ensures efficient resource utilization.

D4—Physical Database Design

PRECEDING ACTIVITY: Performance analysis.

FOLLOWING ACTIVITY: Pseudocode development.

TIME ESTIMATE: One day per table to be designed.

NORMALLY EXECUTED ONCE OR MULTIPLE TIMES: Once per table.

SPECIAL CONSIDERATIONS: If the input to this step is incorrect or ambiguous, the amount of work required here can be much more than that estimated.

DELIVERABLE: Tables, databases physically designed.

Now the output from D3 and/or D4 is used to produce a physical database design. Some of the characteristics of the output include the following:

- Indexing
- Physical attribution of data
- Partitioning strategies
- Designation of keys
- Clustering/interleaving
- Management of variable-length data
- NULL/NOT NULL specification
- Referential integrity

The output of this step is the actual specification of the database to the DBMS or whatever data management software/conventions are adopted.

PARAMETERS OF SUCCESS: Done properly, this stage of analysis transforms all the considerations of logical design of data, performance, update, access, availability, reorganization, restructuring of data, and so on, into a workable database design.

P1—Functional Decomposition

PRECEDING ACTIVITY: Requirements formalization.

FOLLOWING ACTIVITY: Context level 0 specification.

TIME ESTIMATE: Depends on the size of the system and degree of ambiguity (and how firmly and unambiguously the scope has been established). As a rule, two weeks for a reasonably sized system is adequate.

NORMALLY EXECUTED ONCE OR MULTIPLE TIMES: Once per phase of development.

From the requirements document comes the functional decomposition. The functional decomposition merely takes the broad function accomplished by the system and breaks it down into a series of successively smaller functions (down to a level sometimes called the primitive level).

The output of this process is a large functional decomposition describing the different activities to be performed from a high level to a low level.

PARAMETERS OF SUCCESS: Solely reflected in this section are the functions to be designed. Factored into the step are the considerations of other functions that have served or will serve as building blocks. Done properly, this step produces a document that is comprehensible, organized, readable, and complete.

P2—Context Level 0

PRECEDING ACTIVITY: Functional decomposition.

FOLLOWING ACTIVITY: Context level 1-*n* specification.

TIME ESTIMATE: Three days.

NORMALLY EXECUTED ONCE OR MULTIPLE TIMES: Once per phase.

Context level 0 of the functional decomposition describes, at the highest level of abstraction, the major activities to be developed. Context level 0 for process specification corresponds to the ERD in data modeling.

PARAMETERS OF SUCCESS: Done correctly, only the major activities of the system and how they relate are identified in this document.

P3—Context Level 1-*n*

PRECEDING ACTIVITY: Context level 0 specification.

FOLLOWING ACTIVITY: DFD specification.

TIME ESTIMATE: One day per function.

NORMALLY EXECUTED ONCE OR MULTIPLE TIMES: Once per function.

DELIVERABLE: Complete functional decomposition. (Note: Multiple steps contribute to this deliverable.)

The remaining levels of the functional decomposition describe the more detailed activities that occur. Context levels 1-*n* correspond, in terms of process design, to the data item set (dis) in terms of data design.

PARAMETERS OF SUCCESS: When this step has been done properly, all major activities are broken down to a primitive level. The breakdown is orderly, organized, complete, and in accordance with the flow of activity.

P4—Data Flow Diagram (dfd)

PRECEDING ACTIVITY: Context level 1-*n* specification.

FOLLOWING ACTIVITY: Algorithmic specification.

TIME ESTIMATE: One hour per activity (at a primitive level).

NORMALLY EXECUTED ONCE OR MULTIPLE TIMES: Once per activity.

DELIVERABLE: Data flow diagram for each process.

At each context level *n*-the primitive level-a dfd is drawn. The dfd indicates the input to a process, the output of a process, the data stores needed to establish the process, and a brief description of the process. A dfd may be done for context levels higher than *n* if it turns out that a program or process is likely to be written for that context level.

PARAMETERS OF SUCCESS: The input and output for every primitive activity is identified, and the flow of activity is identified as well. Data stores are outlined, and the work to be done by each activity is specified.

P5—Algorithmic Specification; Performance Analysis

PRECEDING ACTIVITY: DFD specification.

FOLLOWING ACTIVITY: Pseudocode.

TIME ESTIMATE: Varies from five minutes to two days per activity at the primitive level that must be specified.

NORMALLY EXECUTED ONCE OR MULTIPLE TIMES: Once per activity.

The processes that are defined by the dfd for primitives are further broken down into detailed algorithmic specification. In other words, in this step, the actual processing to occur step by step is outlined.

In addition, if performance is to be an issue, the effect of performance on program design is factored in. Such design techniques as the following are considered here:

- Breaking a long-running program into a series of shorter programs
- Requiring a program to access a smaller amount of data
- Shortening the time a unit of data is locked
- Changing a lock from update potential to access only

PARAMETERS OF SUCCESS: Algorithmic specification is done correctly when all details needed for specification are identified, and only details needed for specification are identified, one step at a time, including all possibilities. In addition, conformance to the standard work unit (SWU) is ensured here.

P6—Pseudocode

PRECEDING ACTIVITY: Algorithmic specification; physical database design; data store design.

FOLLOWING ACTIVITY: Coding.

TIME ESTIMATE: Varies (see previous activity).

NORMALLY EXECUTED ONCE OR MULTIPLE TIMES: Once per activity (at the primitive level).

The algorithms and program specifications are further refined into pseudocode. The designer ensures that all needed data for processing is available. All variables used in calculations, transformations, updates, and so on are identified here. Any loose ends are identified. Performance at the design level is considered here. The output of this activity is coding specifications ready for actual code.

PARAMETERS OF SUCCESS: The final pass at programming specification includes the following:

- Completeness
- Order of execution
- All required cases
- All contingencies, including error handling and exception conditions
- Structure of coding

P7—Coding

PRECEDING ACTIVITY: Pseudocode.

FOLLOWING ACTIVITY: Walkthrough.

TIME ESTIMATE: Depends, from one day per activity to two weeks per activity.

NORMALLY EXECUTED ONCE OR MULTIPLE TIMES: Once per activity.

DELIVERABLE: Source code.

The pseudocode is translated into source code. If the data has been designed properly, and if the specification of pseudocode has been thorough, this step goes smoothly. The output of this step is source code.

PARAMETERS OF SUCCESS: The complete and efficient translation of pseudocode into code, including inline documentation. Done properly, all requirements previously identified are satisfied by this point.

P8—Walkthrough

PRECEDING ACTIVITY: Coding.

FOLLOWING ACTIVITY: Compilation.

TIME ESTIMATE: One hour per activity.

NORMALLY EXECUTED ONCE OR MULTIPLE TIMES: Once per activity.

The walkthrough is the verbal explanation of code in front of peers. The intent is to find coding errors (or any other kind) before testing. The output of this step is code that has been publicly aired and is as free from error as possible.

PARAMETERS OF SUCCESS: The detection of errors prior to coding. When this step has been done properly, there will be very few errors left to be found by other means.

P9—Compilation

PRECEDING ACTIVITY: Walkthrough.

FOLLOWING ACTIVITY: Unit testing; stress testing.

TIME ESTIMATE: One hour or less per activity.

NORMALLY EXECUTED ONCE OR MULTIPLE TIMES: Until a clean compile is achieved.

Source code is run through the compiler. All errors found in compilation are corrected. The output of this step is compiled code, ready to be tested.

PARAMETERS OF SUCCESS: Compiled code that is ready for testing.

P10—Unit Testing

PRECEDING ACTIVITY: Compilation.

FOLLOWING ACTIVITY: Implementation.

TIME ESTIMATE: Varies widely.

NORMALLY EXECUTED ONCE OR MULTIPLE TIMES: Varies.

Compiled code is tested. There are several levels of unit testing. The simplest (lowest) level of unit testing occurs when the compiled module is tested by itself, ensuring that the function it does is satisfactory. Next, the compiled code is added to other code with which the compiled code will have to work. New levels of unit testing occur in order to ensure the integration of the compiled code with other modules that will be interfaced. A third level of unit testing occurs when major groups of modules are ready to be tested together.

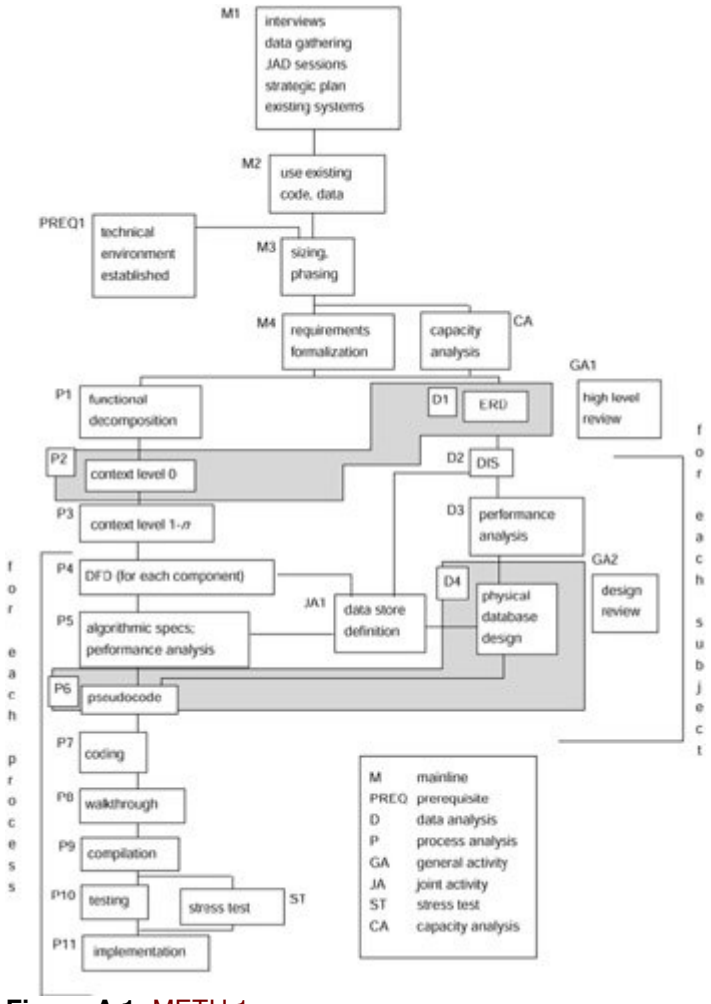

Figure A.1: METH 1.

The output of this step is tested code, ready for execution.

PARAMETERS OF SUCCESS: When executed properly, the code that passes on to the next step has program logic correctly specified. Furthermore, all conditions are tested before code is passed to the next stage, including error and exception conditions.

P11—Implementation

PRECEDING ACTIVITY: Unit test; stress test.

FOLLOWING ACTIVITY: NA

TIME ESTIMATE: NA

DELIVERABLE: A system that satisfies specifications.

There are many activities in implementation. To some extent, implementation is an ongoing activity. Some of the typical activities of implementation are these:

- Training, indoctrination
- Loading of programs
- Initial loading of data

- Conversions of data, if necessary
- Monitoring utilities established
- Writing documentation
- Recovery, reorg procedures established

The output of this step (if there really is an end to the step) is a satisfactorily running system.

PARAMETERS OF SUCCESS: When this step is done, the result is a happy user.

DATA WAREHOUSE DEVELOPMENT—METH 2

DSS1—Data Model Analysis

PRECEDING ACTIVITY: The commitment to build a data warehouse.

FOLLOWING ACTIVITY: Subject area analysis; breadbox analysis; data warehouse design.

TIME ESTIMATE: Varies widely, depending on the status and quality of the data model.

NORMALLY EXECUTED ONCE OR MULTIPLE TIMES: Once.

At the outset, a data model needs to have been defined. The data model needs to have done the following:

- Identified the major subject areas
- Clearly defined boundaries of the model
- Separated primitive from derived data

The following need to be identified for each subject area:

- Keys
- Attributes
- Groupings of attributes
- Relationships among groupings of attributes
- Multiply occurring data
- “Type of” data

The output from this step is a confirmation that the organization has built a solid data model. If the model does not meet the criteria specified, then progress should be halted until the model is brought up to standards of quality.

PARAMETERS OF SUCCESS: The data model will have the following:

- Major subjects identified
- Each major subject with its own separate definition of data, including:
  - Subtypes of data
  - Attributes of data
  - Clearly defined relationships of data
  - Defined groupings of data
  - Defined keys

In addition, each group of data that will go into the data warehouse will have DSS data and operational-only data delineated. All DSS data will have its own time-variant key specified, usually as the lower order of a higher key.

DSS2—Breadbox Analysis

PRECEDING ACTIVITY: Data model analysis.

FOLLOWING ACTIVITY: Data warehouse database design.

TIME ESTIMATE: From one day to two weeks, depending on how well the scope has been defined, how well the data model has been defined, etc.

NORMALLY EXECUTED ONCE OR MULTIPLE TIMES: Once.

DELIVERABLE: Granularity analysis.

Once the model has been analyzed and brought up to a level of sufficient quality, the next step is to do breadbox analysis. *Breadbox analysis* is a sizing-in terms of gross estimates-of the DSS environment. If volume of data is going to be a problem, it is important to know that at the outset. Breadbox analysis simply projects-in raw terms-how much data the data warehouse will hold.

The output of breadbox analysis is simple-if the data warehouse is to contain large amounts of data, then multiple levels of granularity need to be considered. If the data warehouse is not to contain a massive amount of data, then there is no need to plan the design for multiple levels of granularity.

PARAMETERS OF SUCCESS: An estimate of the amount of data-in terms of number of rows-both on the one-year horizon and on the five-year horizon for the entire data warehouse environment, is the result of the process. Based on the results of the estimate, the issue of whether different levels of granularity are needed is decided. If multiple levels of granularity are needed, defining exactly what those levels are is a part of the output of this step.

DSS3—Technical Assessment

PRECEDING ACTIVITY: The commitment to build a data warehouse.

FOLLOWING ACTIVITY: Technical environment preparation.

TIME ESTIMATE: One week.

NORMALLY EXECUTED ONCE OR MULTIPLE TIMES: Once.

DELIVERABLE: Technical environment assessment.

The technical requirements for managing the data warehouse are very different from the technical requirements and consideration for managing data and processing in the operational environment. That is why a separate, central store of DSS data is so popular.

PARAMETERS OF SUCCESS: When executed properly, the technical definition of the data warehouse satisfies the following criteria:

- Ability to manage large amounts of data
- Ability to allow data to be accessed flexibly
- Ability to organize data according to a data model
- Ability both to receive and to send data to a wide variety of technologies
- Ability to have data periodically loaded en masse
- Ability to access data a set at a time or a record at a time

DSS4—Technical Environment Preparation

PRECEDING ACTIVITY: Technical assessment.

FOLLOWING ACTIVITY: Data warehouse design; population.

TIME ESTIMATE: One week to one month.

NORMALLY EXECUTED ONCE OR MULTIPLE TIMES: Once.

Once the architectural configuration for the data warehouse has been established, the next step is to technically identify how the configuration can be accommodated. Some of the typical issues that must be addressed here are the following:

- The amount of DASD required
- What link-either across the network or into the network-will be required
- The volume of processing anticipated
- How to minimize and/or alleviate conflicts of processing between competing access programs
- The volume of traffic that will be generated from the technology that controls the data warehouse
- The nature of traffic-either short or long bursts-generated from the technology that controls the data warehouse

PARAMETERS OF SUCCESS: When this step has been done properly, there are no technical barriers to success. The technical components that should have been installed, allocated, "burned in," and ready to receive data include the following:

- The network
- DASD
- The operating system managing DASD
- The interface to and from the warehouse
- The software used to manage the data warehouse
- The data warehouse

DSS5—Subject Area Analysis

PRECEDING ACTIVITY: Data model analysis.

FOLLOWING ACTIVITY: Source system analysis.

TIME ESTIMATE: One day.

NORMALLY EXECUTED ONCE OR MULTIPLE TIMES: Once per population project.

DELIVERABLE: Which subject area to build next.

Now the subject area to be populated is selected. The first subject area to be selected must be large enough to be meaningful and small enough to be implemented. If by some chance a subject area is truly large and complex, a subset of the subject area may be chosen for implementation. The output from this step is a scope of effort in terms of a subject.

PARAMETERS OF SUCCESS: The output from this phase when done correctly is a definition of what data is to be populated next. In the first few populations, small subjects are usually selected. In later populations, larger subjects, or even subsets of subjects, are selected. When properly done, the subject selected for population meets the needs of the current stage of development of the data warehouse.

DSS6—Data Warehouse Design

PRECEDING ACTIVITY: Data model analysis; source system analysis; breadbox analysis.

FOLLOWING ACTIVITY: Program specification.

TIME ESTIMATE: One week to three weeks.

NORMALLY EXECUTED ONCE OR MULTIPLE TIMES: Once.

DELIVERABLE: Physical database design for the warehouse.

The data warehouse is designed based on the data model. Some of the characteristics of the ultimate design include the following:

- An accommodation of the different levels of granularity, if indeed there are multiple levels of granularity
- An orientation of data to the major subjects of the corporation
- The presence of only primitive data and publicly derived data
- The absence of non-DSS data
- Time variancy of every record of data
- Physical denormalization of data where applicable (i.e., where performance warrants)
- Creation of data artifacts where data that once was in the operational environment is brought over to the data warehouse

The output of this step is a physical database design of the data warehouse. Note that not all of the data warehouse needs to be designed in detail at the outset. It is entirely acceptable to design the major structures of the data warehouse initially, then fill in the details at a later point in time.

PARAMETERS OF SUCCESS: When this step is properly done, the result is a data warehouse that has a manageable amount of data that can be loaded, accessed, indexed, and searched in a reasonably efficient fashion.

DSS7—Source System Analysis

PRECEDING ACTIVITY: Subject area analysis.

FOLLOWING ACTIVITY: Program specification; data warehouse design.

TIME ESTIMATE: One week per subject area.

NORMALLY EXECUTED ONCE OR MULTIPLE TIMES: Once per subject area.

DELIVERABLE: Identification of the system of record.

Once the subject to be populated is identified, the next activity is to identify the source data for the subject in the existing systems environment. It is absolutely normal for there to be a variety of sources of data for DSS data. It is at this point that the issues of integration are addressed. The following represents the issues to be addressed here:

- Key structure/key resolution as data passes from the operational environment to the DSS environment
- Attribution
  - What to do when there are multiple sources to choose from
  - What to do when there are no sources to choose from
  - What transformations-encoding/decoding, conversions, etc.-must be made as data is selected for transport to the DSS environment
- How time variancy will be created from current value data
- Structure-how the DSS structure will be created from the operational structure
- Relationships-how operational relationships will appear in the DSS environment

- The output of this step is the mapping of data from the operational environment to the DSS environment.

PARAMETERS OF SUCCESS: When done properly, the source system that serves the needs of the data warehouse uses data that is timely, complete, accurate, near to the source, easy to access, and that conforms to the structure of data warehouse needs.

DSS8—Specifications

PRECEDING ACTIVITY: Source system analysis; data warehouse design.

FOLLOWING ACTIVITY: Programming.

TIME ESTIMATE: One week per extract/integration program.

NORMALLY EXECUTED ONCE OR MULTIPLE TIMES: Once for each program that needs to be written.

Once the interface between the operational and the DSS environments has been outlined, the next step is to formalize it in terms of program specifications. Some of the major issues here include the following:

- How do I know what operational data to scan?
  - Is the operational data time stamped?
  - Is there a delta file?
  - Are there system logs/audit logs that can be used?
  - Can existing source code and data structure be changed to create a delta file?
  - Do before and after image files have to be rubbed together?
- How do I store the output, once scanned?
  - Is the DSS data preallocated, preformatted?
  - Is data appended?
  - Is data replaced?
  - Are updates in the DSS environment made?

The output from this step is the actual program specifications that will be used to bring data over from the operational environment to the data warehouse.

PARAMETERS OF SUCCESS: When properly done, this step allows the extract and integration of data to be done as efficiently and as simply as possible. (Seldom is this a smooth, simple process.)

DSS9—Programming

PRECEDING ACTIVITY: Specification.

FOLLOWING ACTIVITY: Population.

TIME ESTIMATE: One week per extract/integration program.

NORMALLY EXECUTED ONCE OR MULTIPLE TIMES: Once per program.

DELIVERABLE: Extract, integration, time-perspective program transformation.

This step includes all the standard activities of programming, such as the following:

- Development of pseudocode

- Coding
- Compilation
- Walkthroughs
- Testing-unit, stress-in its many forms

PARAMETERS OF SUCCESS: When done properly, code that is generated out of this step is efficient, documented, able to be changed easily, accurate, and complete.

DSS10—Population

PRECEDING ACTIVITY: Programming; technical environment preparation.

FOLLOWING ACTIVITY: Usage of the data warehouse.

TIME ESTIMATE: NA

NORMALLY EXECUTED ONCE OR MULTIPLE TIMES: NA

DELIVERABLE: Usable data warehouse.

This step entails nothing more than the execution of the DSS programs previously developed. The issues addressed here are the following:

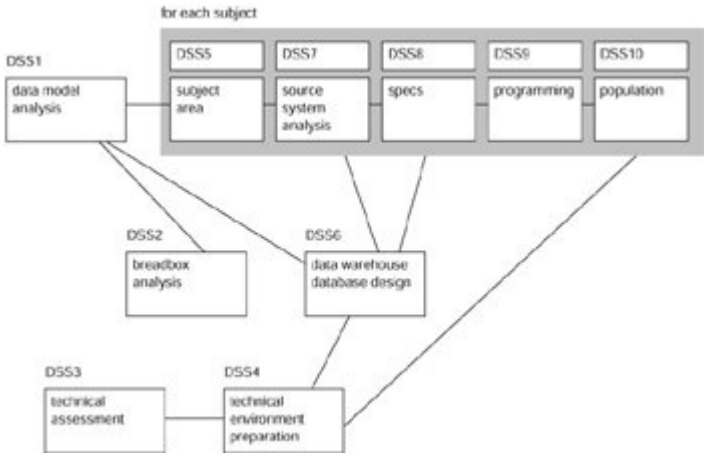

Figure A.2: METH 2.

- Frequency of population
- Purging populated data
- Aging populated data (i.e., running tallying summary programs)
- Managing multiple levels of granularity
- Refreshing living sample data (if living sample tables have been built)

The output of this step is a populated, functional data warehouse.

PARAMETERS OF SUCCESS: When done properly, the result is an accessible, comprehensible warehouse that serves the needs of the DSS community.

HEURISTIC PROCESSING—METH 3

The third phase of development in the architected environment is the usage of data warehouse data for the purpose of analysis. Once the data in the data warehouse environment is populated, usage may commence.

There are several essential differences between the development that occurs at this level and development in other parts of the environment. The first major difference is that at this phase the development process always starts with data, that is, the data in the data warehouse. The second difference is that requirements are not known at the start of the development process. The third difference (which is really a byproduct of the first two factors) is that processing is done in a very iterative, heuristic fashion. In other types of development, there is always a certain amount of iteration. But in the DSS component of development that occurs after the data warehouse is developed, the whole nature of iteration changes. Iteration of processing is a normal and essential part of the analytical development process, much more so than it is elsewhere.

The steps taken in the DSS development components can be divided into two categories-the repetitively occurring analysis (sometimes called the “departmental” or “functional” analysis) and the true heuristic processing (the “individual” level).

Figure A.3 shows the steps of development to be taken after the data warehouse has begun to be populated.

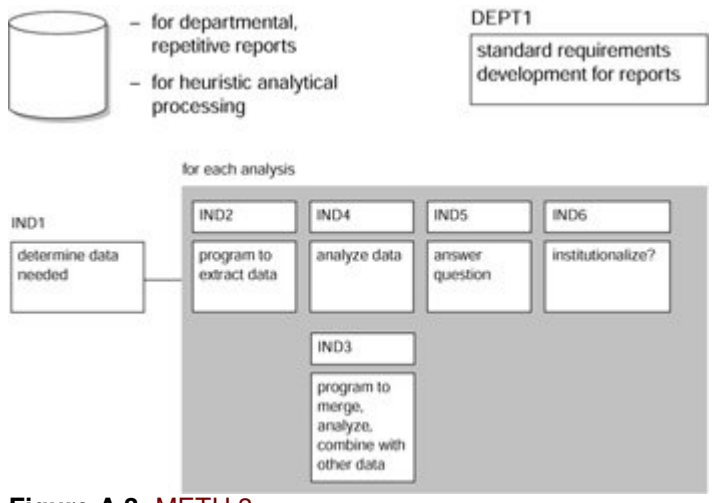

Figure A.3: METH 3.

HEURISTIC DSS DEVELOPMENT—METH 4

DEPT1-Repeat Standard Development-For repetitive analytical processing (usually called delivering standard reports), the normal requirements-driven processing occurs. This means that the following steps (described earlier) are repeated:

M1—interviews, data gathering, JAD, strategic plan, existing systems

M2—sizing, phasing

M3—requirements formalization

P1—functional decomposition

P2—context level 0

P3—context level 1-n

P4—dfd for each component

P5—algorithmic specification; performance analysis

P6—pseudocode

P7—coding

P8—walkthrough

P9—compilation

P10—testing

P11—implementation

In addition, at least part of the following will occur at the appropriate time:

GA1—high-level review

GA2—design review

It does not make sense to do the data analysis component of development because the developer is working from the data warehouse.

The output of this activity are reports that are produced on a regular basis.

PARAMETERS OF SUCCESS: When done properly, this step ensures that regular report needs are met. These needs usually include the following:

- Regulatory reports
- Accounting reports
- Key factor indicator reports
- Marketing reports
- Sales reports

Information needs that are predictable and repetitive are met by this function.

NOTE: For highly iterative processing, there are parameters of success, but they are met collectively by the process. Because requirements are not defined a priori, the parameters of success for each iteration are somewhat subjective.

IND1—Determine Data Needed

At this point, data in the data warehouse is selected for potential usage in the satisfaction of reporting requirements. While the developer works from an educated-guess perspective, it is understood that the first two or three times this activity is initiated, only some of the needed data will be retrieved.

The output from this activity is data selected for further analysis.

IND2—Program to Extract Data

Once the data for analytical processing is selected, the next step is to write a program to access and strip the data. The program written should be able to be modified easily because it is anticipated that the program will be run, modified, then rerun on numerous occasions.

DELIVERABLE: Data pulled from the warehouse for DSS analysis.

IND3—Combine, Merge, Analyze

After data has been selected, it is prepared for analysis. Often this means editing the data, combining it with other data, and refining it.

Like all other heuristic processes, it is anticipated that this program be written so that it is easily modifiable and able to be rerun quickly. The output of this activity is data fully usable for analysis.

DELIVERABLE: Analysis with other relevant data.

IND4—Analyze Data

Once data has been selected and prepared, the question is “Do the results obtained meet the needs of the analyst?” If the results are not met, another iteration occurs. If the results are met, then the final report preparation is begun.

DELIVERABLE: Fulfilled requirements.

IND5—Answer Question

The final report that is produced is often the result of many iterations of processing. Very seldom is the final conclusion the result of a single iteration of analysis.

IND6—Institutionalization

The final issue to be decided is whether the final report that has been created should be institutionalized. If there is a need to run the report repetitively, it makes sense to submit the report as a set of requirements and to rebuild the report as a regularly occurring operation.

Summary

How the different activities relate to each other and to the notion of data architecture are described by the diagram shown in Figure A.4.

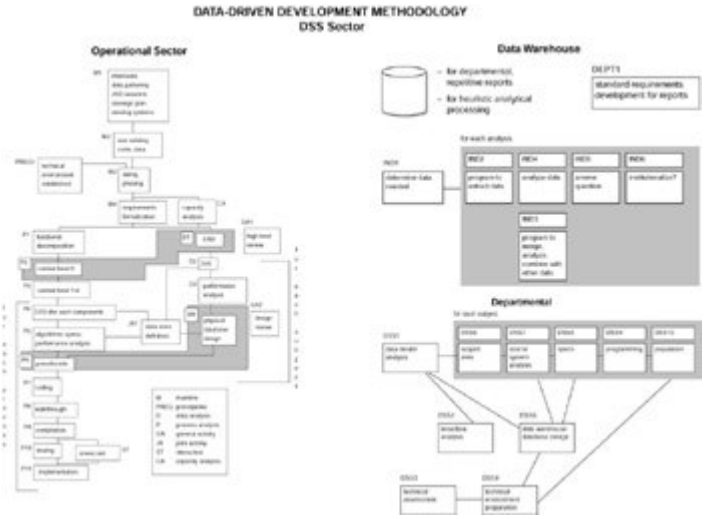

Figure A.4: METH 4. Data-Driven development methodology.

Selected Topics

The best way to describe the data-driven nature of the development methodology is graphically. Figure A.5 shows that the data model is at the heart of the data-driven methodology.

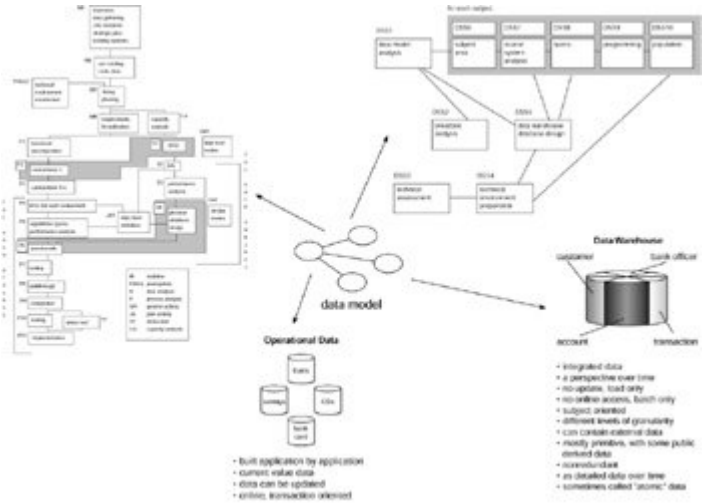

Figure A.5: METH 5.

The data model relates to the design of operational data, to the design of data in the data warehouse, to the development and design process for operational data, and to the development and design process for the data warehouse. Figure A.5 shows how the same data model relates to each of those activities and databases.

The data model is the key to identifying commonality across applications. But one might ask, “Isn’t it important to recognize the commonality of processing as well?”

The answer is that, of course, it is important to recognize the commonality of processing across applications. But there are several problems with trying to focus on the commonality of processes-processes change much more rapidly than data, processes tend to mix common and unique processing so tightly that they are often inseparable, and classical process analysis often places an artificially small boundary on the scope of the design. Data is inherently more stable than processing. The scope of a data analysis is easier to enlarge than the scope of a process model. Therefore, focusing on data as the keystone for recognizing commonality makes sense. In addition, the assumption is made that if commonality of data is discovered, the discovery will lead to a corresponding commonality of processing.

For these reasons, the data model-which cuts across all applications and reflects the corporate perspective-is the foundation for identifying and unifying commonality of data and processing.

Deliverables

The steps of the data-driven development methodology include a deliverable. In truth, some steps contribute to a deliverable with other steps. For the most part, however, each step of the methodology has its own unique deliverable.

The deliverables of the process analysis component of the development of operational systems are shown by Figure A.6.

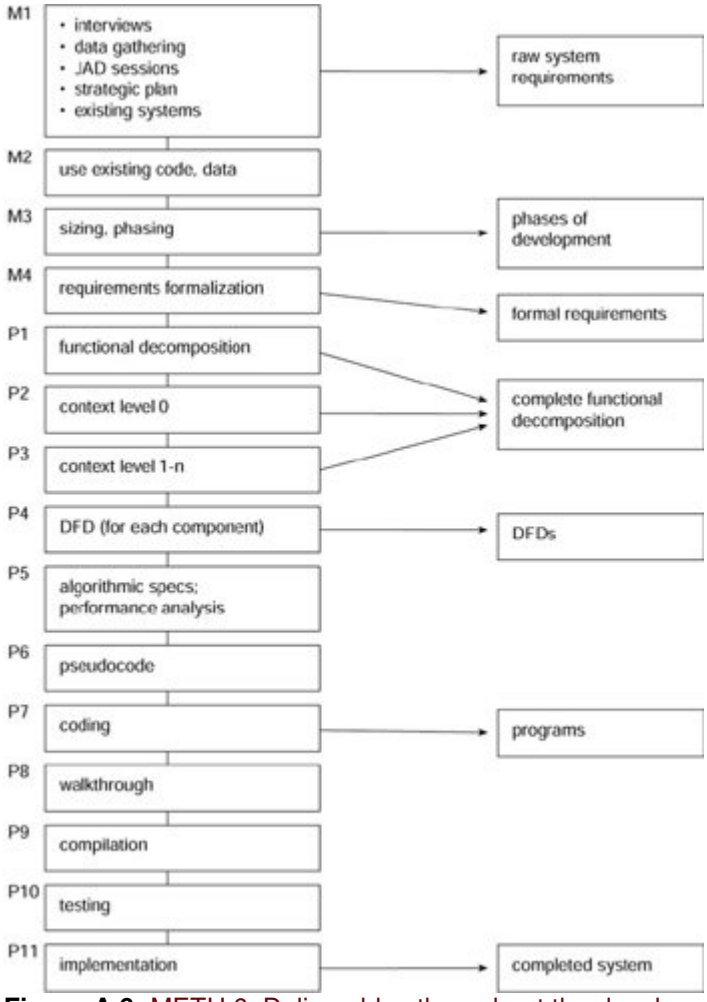

Figure A.6: METH 6. Deliverables throughout the development life cycle.

Figure A.6 shows that the deliverable for the interview and data-gathering process is a raw set of systems requirements. The analysis to determine what code/data can be reused and the step for sizing/phasing the raw requirements contribute a deliverable describing the phases of development.

The activity of requirements formalization produces (not surprisingly) a formal set of system specifications. The result of the functional decomposition activities is the deliverable of a complete functional decomposition.

The deliverable for the dfd definition is a set of dfds that describe the functions that have been decomposed. In general, the dfds represent the primitive level of decomposition.

The activity of coding produces the deliverable of programs. And finally, the activity of implementation produces a completed system.

The deliverables for data analysis for operational systems are shown in Figure A.7.

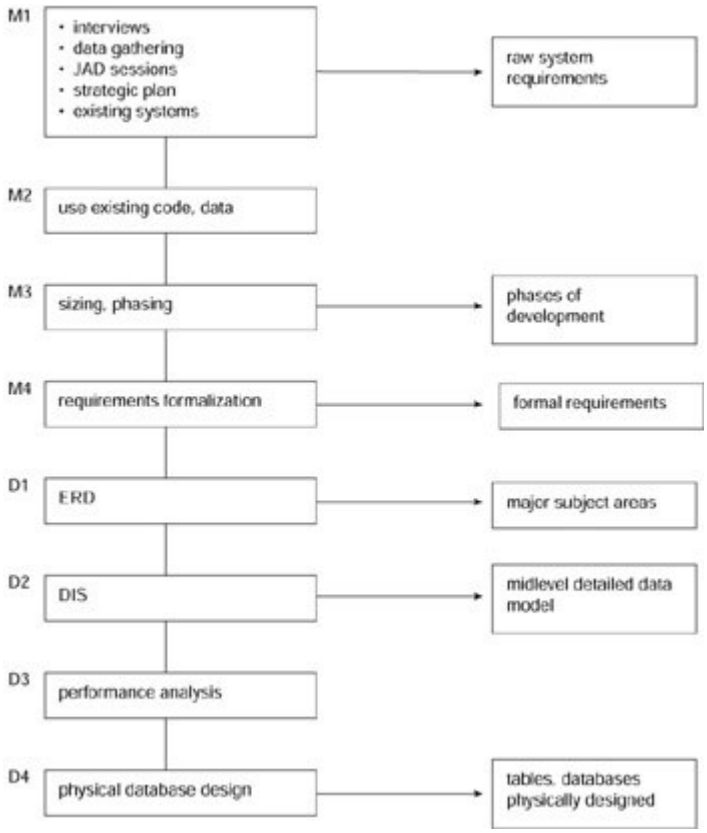

Figure A.7: METH 7. Deliverables for operational data analysis.

The same deliverables discussed earlier are produced by the interview and data gathering process, the sizing and phasing activity, and the definition of formal requirements.

The deliverable of the ERD activity is the identification of the major subject areas and their relationship to each other. The deliverable of the dis activity is the fully attributed and normalized description of each subject area. The final deliverable of physical database design is the actual table or database design, ready to be defined to the database management system(s).

The deliverables of the data warehouse development effort are shown in Figure A.8, where the result of the breadbox analysis is the granularity and volume analysis. The deliverable associated with data warehouse database design is the physical design of data warehouse tables. The deliverable associated with technical environment preparation is the establishment of the technical environment in which the data warehouse will exist. Note that this environment may or may not be the same environment in which operational systems exist.

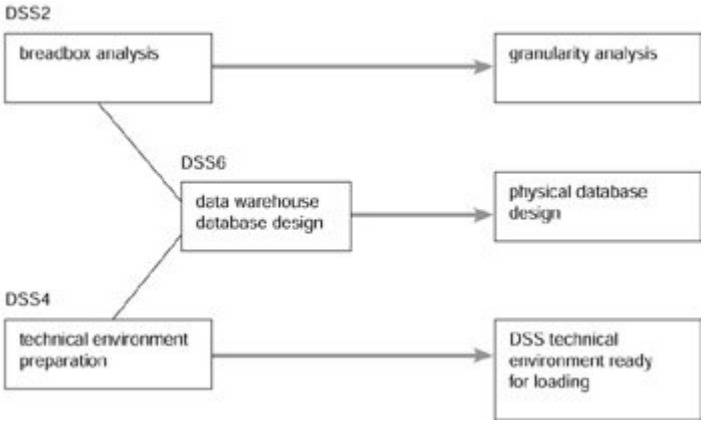

Figure A.8: METH 8. Preliminary data warehouse deliverables.

On a repetitive basis, the deliverables of data warehouse population activities are represented by Figure A.9, which shows that the deliverable for subject area analysis-each time the data warehouse is to be populated-is the selection of a subject (or possibly a subset of a subject) for population.

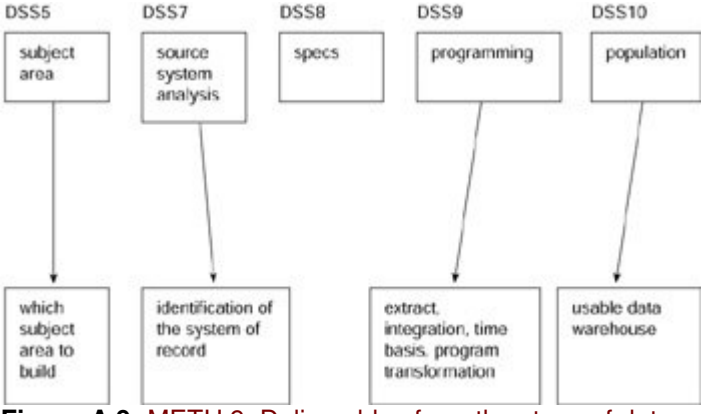

Figure A.9: METH 9. Deliverables from the steps of data warehouse development.

The deliverable for source system analysis is the identification of the system of record for the subject area being considered. The deliverable for the programming phase is the programs that will extract, integrate, and change data from current value to time variant.

The final deliverable in the population of the data warehouse is the actual population of the warehouse. It is noted that the population of data into the warehouse is an ongoing activity.

Deliverables for the heuristic levels of processing are not as easy to define as they are for the operational and data warehouse levels of development. The heuristic nature of the analytical processing in this phase is much more informal. However, Figure A.10 shows some of the deliverables associated with heuristic processing based on the data warehouse.

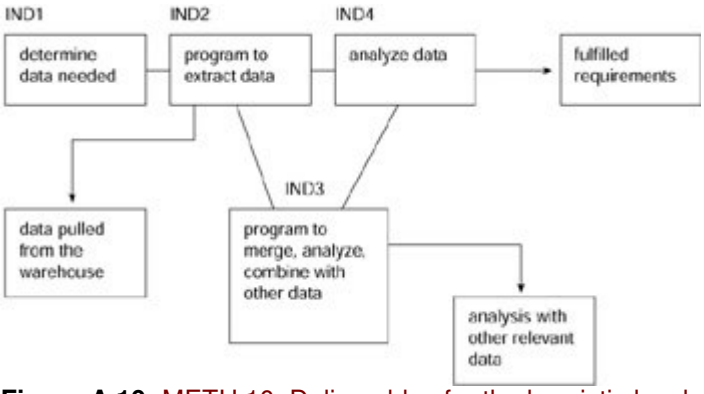

Figure A.10: METH 10. Deliverables for the heuristic level of processing.

Figure A.10 shows that data pulled from the warehouse is the result of the extraction program. The deliverable of the subsequent analysis step is further analysis based on data already refined. The deliverable of the final analysis of data is the satisfaction (and understanding) of requirements.

A Linear Flow of Deliverables

Except for heuristic processing, a linear flow of deliverables is to be expected. Figure A.11 shows a sample of deliverables that would result from the execution of the process analysis component of the data-driven development methodology.

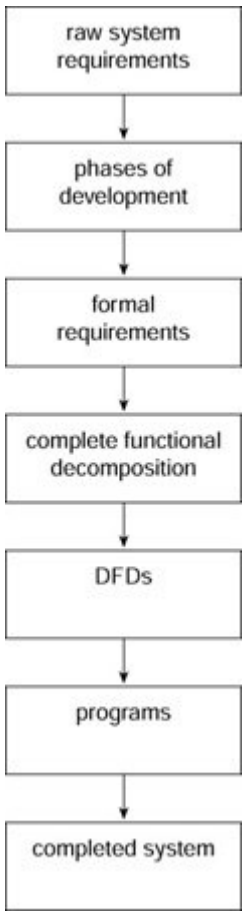

Figure A.11: METH 11. A linear flow of deliverables for operational process analysis.

It is true that within reason there is a linear flow of deliverables; however, the linear flow shown glosses over two important aspects:

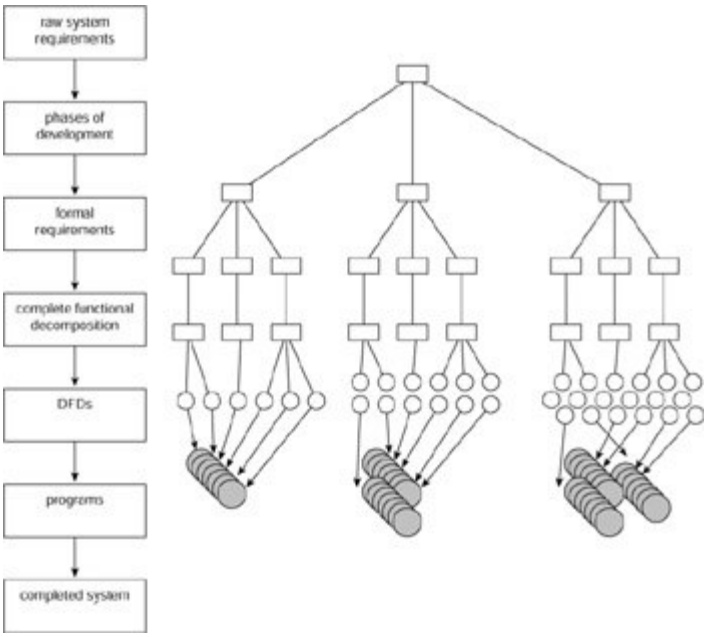

Figure A.12: METH 12. Deliverables usually spawn multiple deliverables at a lower level.

- The deliverables are usually produced in an iterative fashion.
- There are multiple deliverables at any given level. In other words, deliverables at any one level have the capability of spawning multiple deliverables at the next lower level, as shown by Figure A.12.

Figure A.12 shows that a single requirements definition results in three development phases. Each development phase goes through formal requirements definition and into decomposition. From the decomposition, multiple activities are identified, each of which has a dfd created for it. In turn, each dfd creates one or more programs. Ultimately, the programs form the backbone of the completed system.

Estimating Resources Required for Development

Looking at the diagram shown in Figure A.12, it becomes apparent that once the specifics of exactly how many deliverables are being spawned are designed, then an estimation of how many resources the development process will take can be rationally done.

Figure A.13 shows a simple technique, in which each level of deliverables first is defined so that the total number of deliverables is known. Then the time required for the building of each deliverable is multiplied by each deliverable, yielding an estimate of the employee resources required.

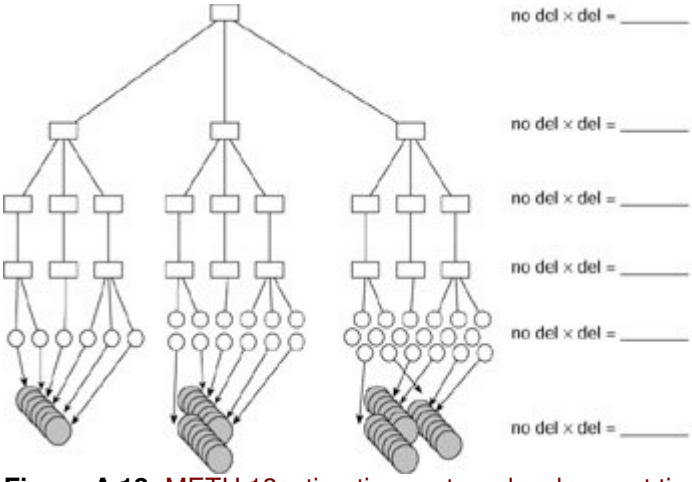

Figure A.13: METH 13. stimating system development time.

SDLC/CLDS

Earlier discussions alluded to the fact that operational systems are built under one system development life cycle, and DSS systems are built under another system development life cycle. Figure A.14 shows the development life cycle associated with operational systems, where requirements are at the starting point. The next activities include analysis, design, programming, testing, integration, implementation, and maintenance.

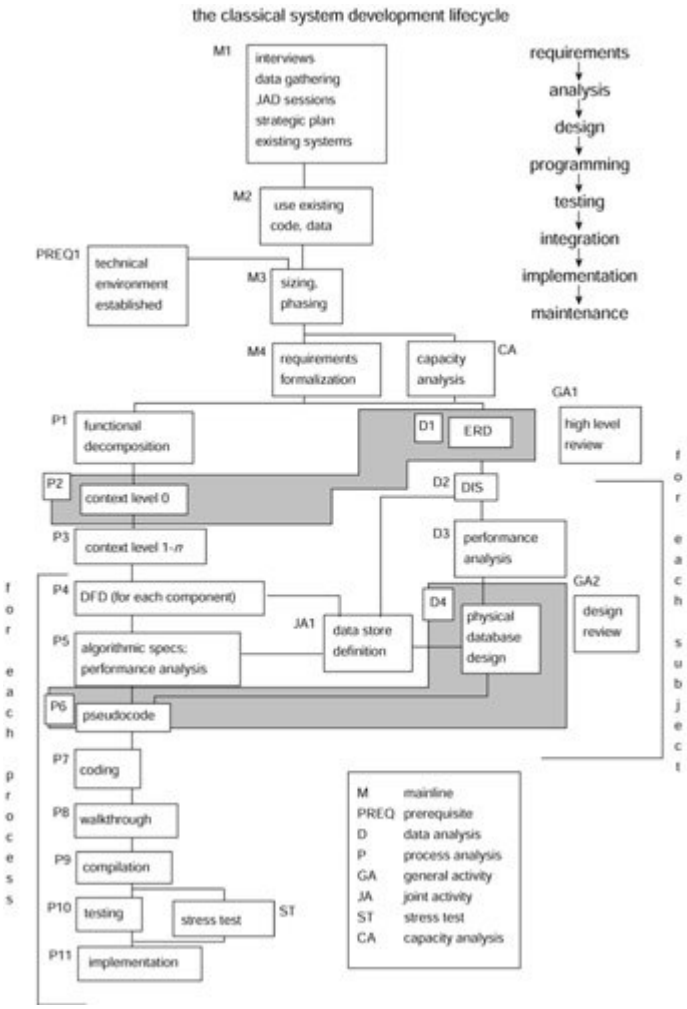

Figure A.14: METH 14.

The system development life cycle associated with DSS systems is shown by Figure A.15, where DSS processing begins with data. Once data for analysis is secured (usually by using the data warehouse), programming, analysis, and so forth continue. The development life cycle for DSS data ends with an understanding of the requirements.

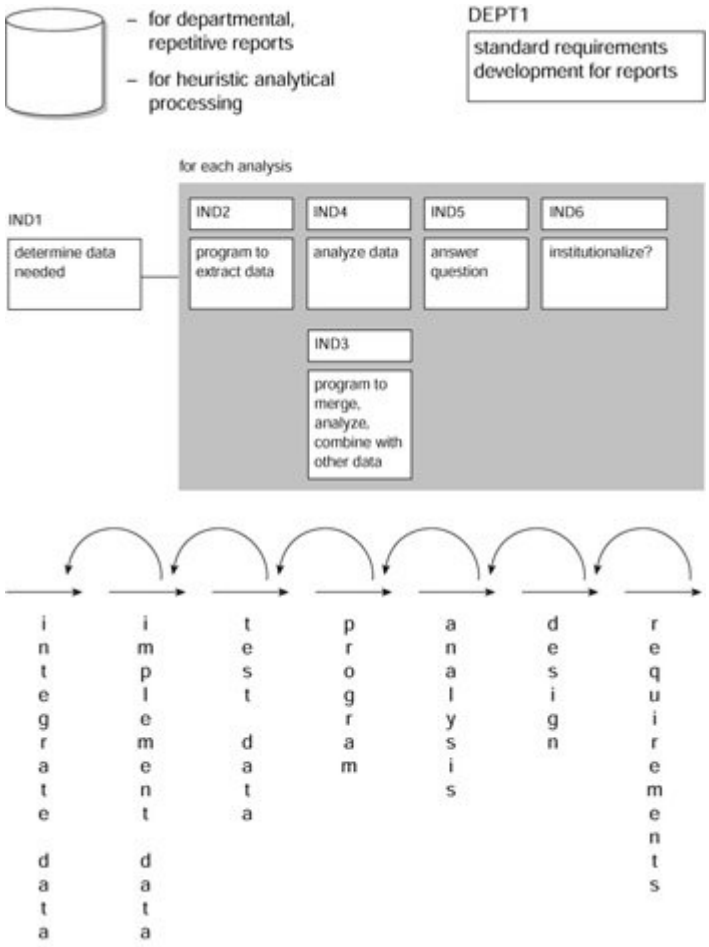

Figure A.15: METH 15.

Data Dictionary

The role of the data dictionary is depicted by the graph shown in Figure A.16.

Figure A.16: METH 16. Data warhouse development.

The data dictionary plays a central role in operational processing in the activities of ERD development and documentation, DIS development, physical database design, and coding. The data dictionary plays a heavy role in data model analysis, subject area selection, source system selection (system of record identification), and programming in the world of data warehouse development.

What about Existing Systems?

In very few cases is development done freshly with no backlog of existing systems. Existing systems certainly present no problem to the DSS component of the data-driven development methodology. Finding the system of record in existing systems to serve as a basis for warehouse data is a normal event.

A word needs to be said about existing systems in the operational environment. The first approach to existing operational systems is to try to build on them. When this is possible, much productivity is the result. But in many cases existing operational systems cannot be built on.

The second stance is to try to modify existing operational systems. In some cases, this is a possibility; in most cases, it is not.

The third stance is to do a wholesale replacement and enhancement of existing operational systems. In this case, the existing operational system serves as a basis for gathering requirements, and no more.

A variant of a wholesale replacement is the conversion of some or all of an existing operational system. This approach works on a limited basis, where the existing system is small and simple. The larger and more complex the existing operational system, the less likelihood that the system can be converted.

Glossary

A

access

the operation of seeking, reading, or writing data on a storage unit.

access method

a technique used to transfer a physical record from or to a mass storage device.

access pattern

the general sequence in which the data structure is accessed (for example, from tuple to tuple, from record to record, from segment to segment, etc.).

accuracy

a qualitative assessment of freedom from error or a quantitative measure of the magnitude of error, expressed as a function of relative error.

ad hoc processing

one-time-only, casual access and manipulation of data on parameters never before used, usually done in a heuristic, iterative manner.

after image

the snapshot of data placed on a log on the completion of a transaction.

agent of change

a motivating force large enough not to be denied, usually aging of systems, changes in technology, radical changes in requirements, etc.

algorithm

a set of statements organized to solve a problem in a finite number of steps.

alternate storage

storage other than disk-based storage used to hold bulk amounts of relatively inactive storage.

analytical processing

using the computer to produce an analysis for management decision, usually involving trend analysis, drill-down analysis, demographic analysis, profiling, etc.

application

a group of algorithms and data interlinked to support an organizational requirement.

application database

a collection of data organized to support a specific application.

archival database

a collection of data containing data of a historical nature. As a rule, archival data cannot be updated. Each unit of archival data is relevant to a moment in time, now passed.

artifact

a design technique used to represent referential integrity in the DSS environment.

atomic

(1) data stored in a data warehouse; (2) the lowest level of process analysis.

atomic database

a database made up of primarily atomic data; a data warehouse; a DSS foundation database.

atomic-level data

data with the lowest level of granularity. Atomic-level data sits in a data warehouse and is time-variant (i.e., accurate as of some moment in time now passed).

attribute

a property that can assume values for entities or relationships. Entities can be assigned several attributes (for example, a tuple in a relationship consists of values). Some systems also allow relationships to have attributes as well.

audit trail

data that is available to trace activity, usually update activity.

B

backup

a file serving as a basis for the activity of backing up a database. Usually a snapshot of a database as of some previous moment in time.

batch

computer environment in which programs (usually long-running, sequentially oriented) access data exclusively, and user interaction is not allowed while the activity is occurring.

batch environment

a sequentially dominated mode of processing; in batch, input is collected and stored for future, later processing. Once collected, the batch input is transacted sequentially against one or more databases.

before image

a snapshot of a record prior to update, usually placed on an activity log.

bitmap

a specialized form of an index indicating the existence or nonexistence of a condition for a group of blocks or records. Bitmaps are expensive to build and maintain but provide very fast comparison and access facilities.

blocking

the combining of two or more physical records so that they are physically located together. The result of their physical colocation is that they can be accessed and fetched by a single execution of a machine instruction.

C

cache

a buffer usually built and maintained at the device level. Retrieving data out of a cache is much quicker than retrieving data out of a cylinder.

cardinality (of a relation)

the number of tuples (i.e., rows) in a relation.

CASE

computer-aided software engineering

checkpoint

an identified snapshot of the database or a point at which the transactions against the database have been frozen or have been quiesced.

checkpoint/restart

a means of restarting a program at some point other than the beginning for example, when a failure or interruption has occurred. *N*

checkpoints may be used at intervals throughout an application program. At each of those points, sufficient information is stored to permit the program to be restored to the moment in time the checkpoint has been taken.

CLDS

the facetiously named system development life cycle for analytical, DSS systems. CLDS is so named because, in fact, it is the reverse of the classical systems development life cycle SDLC.

clickstream data

data generated in the Web environment that tracks the activity of the users of the Web site.

column

a vertical table in which values are selected from the same domain. A row is made up of one or more columns.

Common Business Oriented Language (COBOL)

a computer language for the business world. A very common language.

commonality of data

similar or identical data that occurs in different applications or systems. The recognition and management of commonality of data is one of the foundations of conceptual and physical database design.

compaction

a technique for reducing the number of bits required to represent data without losing the content of the data. With compaction, repetitive data are represented very concisely.

condensation

the process of reducing the volume of data managed without reducing the logical consistency of the data. Condensation is essentially different from compaction.

contention

the condition that occurs when two or more programs try to access the same data at the same time.

continuous time span data

data organized so that a continuous definition of data over a span of time is represented by one or more records.

corporate information factory (CIF)

the framework that exists that surrounds the data warehouse; typically contains an ODS, a data warehouse, data marts, DSS applications, exploration warehouses, data mining warehouses, alternate storage, and so forth

CPU

central processing unit.

CPU-bound

the state of processing in which the computer can produce no more output because the CPU portion of the processor is being used at 100 percent capacity. When the computer is CPU-bound, typically the memory and storage processing units are less than 100 percent utilized. With modern DBMS, it is much more likely that the computer is I/O-bound, rather than CPU-bound.

CRM

customer relationship management, a popular DSS application designed to streamline customer/corporate relationships.

cross-media storage manager

software whose purpose is to move data to and from disk storage and alternate storage.

current value data

data whose accuracy is valid as of the moment of execution, as opposed to time-variant data.

D

DASD

see direct access storage device.

data

a recording of facts, concepts, or instructions on a storage medium for communication, retrieval, and processing by automatic means and presentation as information that is understandable by human beings.

data administrator (DA)

the individual or organization responsible for the specification, acquisition, and maintenance of data management software and the design, validation, and security of files or databases. The data model and the data dictionary are classically the charge of the DA.

database

a collection of interrelated data stored (often with controlled, limited redundancy) according to a schema. A database can serve single or multiple applications.

database administrator (DBA)

the organizational function charged with the day-to-day monitoring and care of the databases. The DBA function is more closely associated with physical database design than the DA is.

database key

a unique value that exists for each record in a database. The value is often indexed, although it can be randomized or hashed.

database management system (DBMS)

a computer-based software system used to establish and manage data.

data-driven development

the approach to development that centers around identifying the commonality of data through a data model and building programs that have a broader scope than the immediate application. Data-driven development differs from classical application-oriented development.

data element

(1) an attribute of an entity; (2) a uniquely named and well-defined category of data that consists of data items and that is included in a record of an activity.

data item set (dis)

a grouping of data items, each of which directly relates to the key of the grouping of data in which the data items reside. The data item set is found in the midlevel model.

data mart

a departmentalized structure of data feeding from the data warehouse where data is denormalized based on the department's need for information.

data mining

the process of analyzing large amounts of data in search of previously undiscovered business patterns.

data model

(1) the logical data structures, including operations and constraints provided by a DBMS for effective database processing; (2) the system used for the representation of data (for example, the ERD or relational model).

data structure

a logical relationship among data elements that is designed to support specific data manipulation functions (trees, lists, and tables).

data warehouse

a collection of integrated, subject-oriented databases designed to support the DSS function, where each unit of data is relevant to some moment in time. The data warehouse contains atomic data and lightly summarized data.

decision support system (DSS)

a system used to support managerial decisions. Usually DSS involves the analysis of many units of data in a heuristic fashion. As a rule, DSS processing does not involve the update of data.

decompaction

the opposite of compaction; once data is stored in a compacted form, it must be decompacted to be used.

denormalization

the technique of placing normalized data in a physical location that optimizes the performance of the system.

derived data

data whose existence depends on two or more occurrences of a major subject of the enterprise.

derived data element

a data element that is not necessarily stored but that can be generated when needed (age, current date, date of birth).

design review

the quality assurance process in which all aspects of a system are reviewed publicly prior to the striking of code.

dimension table

the place where extraneous data that relates to a fact table is placed in a multidimensional table.

direct access

retrieval or storage of data by reference to its location on a volume. The access mechanism goes directly to the data in question, as is generally required with online use of data. Also called random access or hashed access.

direct access storage device (DASD)

a data storage unit on which data can be accessed directly without having to progress through a serial file such as a magnetic tape file. A disk unit is a direct access storage device.

dormant data

data that is very infrequently used.

download

the stripping of data from one database to another based on the content of data found in the first database.

drill-down analysis

the type of analysis where examination of a summary number leads to the exploration of the components of the sum.

DSS application

an application whose foundation of data is the data warehouse.

dual database

the practice of separating high-performance, transaction-oriented data from decision support data.

dual database management systems

the practice of using multiple database management systems to control different aspects of the database environment.

dumb terminal

a device used to interact directly with the end user where all processing is done on a remote computer. A dumb terminal acts as a device that gathers data and displays data only.

E

estetbusiness

commerce conducted based on Web interactions.

encoding

a shortening or abbreviation of the physical representation of a data value (e.g., male 5 “M,” female 5 “F”).

enterprise resource planning (ERP)

application software for processing transactions.

entity

a person, place, or thing of interest to the data modeler at the highest level of abstraction.

entity-relationship diagram (ERD)

a high-level data model; the schematic showing all the entities within the scope of integration and the direct relationship between those entities.

event

a signal that an activity of significance has occurred. An event is noted by the information system.

Executive Information Systems (EIS)

systems designed for the top executive, featuring drill-down analysis and trend analysis.

extract/load/transformation (ETL)

the process of taking legacy application data and integrating it into the data warehouse.

external data

(1) data originating from other than the operational systems of a corporation; (2) data residing outside the central processing complex.

exploration warehouse

a structure specifically designed for statistical processing that searches for business patterns.

extract

the process of selecting data from one environment and transporting it to another environment.

F-G

fact table

the center of a star join table where data that has many occurrences will be located.

flat file

a collection of records containing no data aggregates, nested repeated data items, or groups of data items.

foreign key

an attribute that is not a primary key in a relational system but whose values are the values of the primary key of another relation.

fourth-generation language

language or technology designed to allow the end user unfettered access to data.

functional decomposition

the division of operations into hierarchical functions (activities) that form the basis for procedures.

global data warehouse

a warehouse suited to the needs of headquarters of a large corporation.

granularity

the level of detail contained in a unit of data. The more detail there is, the lower the level of granularity. The less detail there is, the higher the level of granularity.

granularity manager

the software or processes that edit and filter Web data as it flows into the data warehouse. The data that flows into the data warehouse environment from the Web environment is usually clickstream data that is stored in a Web log.

H-J

heuristic

the mode of analysis in which the next step is determined by the results of the current step of analysis. Used for decision support processing.

image copy

a procedure in which a database is physically copied to another medium for the purposes of backup.

index

the portion of the storage structure maintained to provide efficient access to a record when its index key item is known.

information

data that human beings assimilate and evaluate to solve a problem or make a decision.

integrity

the property of a database that ensures that the data contained in the database is as accurate and consistent as possible.

interactive

a mode of processing that combines some of the characteristics of online transaction processing and batch processing. In interactive processing, the end user interacts with data over which he or she has exclusive control. In addition, the end user can initiate background activity to be run against the data.

Internet

a network of users who have access to data and Web addresses around the world.

"is a type of"

an analytical tool used in abstracting data during the process of conceptual database design (for example, a cocker spaniel is a type of dog).

iterative analysis

the mode of processing in which the next step of processing depends on the results obtained by the existing step in execution; heuristic processing.

joint application design (JAD)

an organization of people, usually end users, who create and refine application system requirements.

judgment sample

a sample of data where data is accepted or rejected for the sample based on one or more parameters.

K-L

key

a data item or combination of data items used to identify or locate a record instance (or other similar data groupings).

key, primary

a unique attribute used to identify a single record in a database.

key, secondary

a nonunique attribute used to identify a class of records in a database.

living sample

a representative database typically used for heuristic, statistical, analytical processing in place of a large database. Periodically, the very large database is selectively stripped of data so that the resulting living sample database represents a cross-section of the very large database as of some moment in time.

load

to insert data values into a database that was previously empty.

local data warehouse

a data warehouse holding geographically local data in support of a global data warehouse.

log

a journal of activity.

logging

the automatic recording of data with regard to the access of the data, the updates to the data, etc.

loss of identity

when data is brought in from an external source and the identity of the external source is discarded, loss of identity occurs. A common practice with microprocessor data.

M-N

magnetic tape

(1) the storage medium most closely associated with sequential processing; (2) a large ribbon on which magnetic images are stored and retrieved.

master file

a file that holds the system of record for a given set of data (usually bound by an application).

metadata

(1) data about data; (2) the description of the structure, content, keys, indexes, etc., of data.

microprocessor

a small processor serving the needs of a single user.

migration

the process by which frequently used items of data are moved to more readily accessible areas of storage and infrequently used items of data are moved to less readily accessible areas of storage.

million instructions per second (mips)

the standard measurement of processor speed for minicomputers and mainframe computers.

multidimensional processing

data mart processing based on a star join structuring of data.

near line storage

data that is not stored on disk but is never the less still accessible; used to hold very large amounts of relatively inactive data.

O-P

online analytical processing (OLAP)

departmental processing for the data mart environment.

online storage

storage devices and storage media where data can be accessed in a direct fashion.

operational data

data used to support the daily processing a company does.

operational data store (ODS)

a hybrid structure designed to support both operational transaction processing and analytical processing.

operations

the department charged with the running of the computer.

optical disk

a storage medium using lasers as opposed to magnetic devices. Optical disk is typically write only, is much less expensive per byte than magnetic storage, and is highly reliable.

overflow

(1) the condition in which a record or a segment cannot be stored in its home address because the address is already occupied. In this case, the data is placed in another location referred to as overflow; (2) the area of DASD where data is sent when the overflow condition is triggered.

ownership

the responsibility for updating operational data.

page

(1) a basic unit of data on DASD; (2) a basic unit of storage in main memory.

parameter

an elementary data value used as a criterion for qualification, usually of data searches or in the control of modules.

partition

a segmentation technique in which data is divided into physically different units. Partitioning can be done at the application or the system level.

populate

to place occurrences of data values in a previously empty database. See load.

primary key

an attribute that contains values that uniquely identify the record in which the key exists.

primitive data

data whose existence depends on only a single occurrence of a major subject area of the enterprise.

processor

the hardware at the center of execution of computer programs. Generally speaking, processors are divided into three categories: mainframes, minicomputers, and microcomputers.

processor cycles

the hardware's internal cycles that drive the computer (e.g., initiate I/O, perform logic, move data, perform arithmetic functions).

production environment

the environment where operational, high-_performance processing is run.

punched cards

an early storage medium on which data and input were stored. Today punched cards are rare.

Q-S

query language

a language that enables an end user to interact directly with a DBMS to retrieve and possibly modify data held under the DBMS.

record

an aggregation of values of data organized by their relation to a common key.

record-at-a-time processing

the access of data a record at a time, a tuple at a time, etc.

recovery

the restoration of the database to an original position or condition, often after major damage to the physical medium.

redundancy

the practice of storing more than one occurrence of data. In the case where data can be updated, redundancy poses serious problems. In the case where data is not updated, redundancy is often a valuable and necessary design technique.

referential integrity

the facility of a DBMS to ensure the validity of predefined relationships.

reorganization

the process of unloading data in a poorly organized state and reloading the data in a well-organized state. Reorganization in some DBMSs is used to restructure data. Reorganization is often called “reorg,” or an “unload/reload” process.

repeating groups

a collection of data that can occur several times within a given record occurrence.

rolling summary

a form of storing archival data where the most recent data has the most details stored, and data that is older has fewer details stored.

scope of integration

the formal definition of the boundaries of the system being modeled.

SDLC

system development life cycle; the classical operational system development life cycle that typically includes requirements gathering, analysis, design, programming, testing, integration, and implementation.

sequential file

a file in which records are ordered according to the values of one or more key fields. The records can be processed in this sequence starting from the first record in the file, continuing to the last record in the file.

serial file

a sequential file in which the records are physically adjacent, in sequential order.

set-at-a-time processing

access of data by groups, each member of which satisfies a selection criterion.

snapshot

a database dump or the archiving of data out of a database as of some moment in time.

snowflake structure

the result of joining two or more star joins.

spiral development

iterative development, as opposed to waterfall development.

solutions database

the component of a DSS environment where the results of previous decisions are stored. Solutions databases are consulted to help determine the proper course of action in a current decision-making situation.

staging area

a place where data in transit is placed, usually coming from the legacy environment prior to entering the ETL layer of processing.

star join

a data structure where data is denormalized to optimize the access of the data; the basis of multidimensional data mart design.

storage hierarchy

storage units linked to form a storage subsystem, in which some units are fast but small and expensive, and other units are large but slower and less expensive.

subject database

a database organized around a major subject of the corporation. Classical subject databases are for customer, transaction, product, part, vendor.

system log

an audit trail of relevant system events (for example, transaction entries, database changes, etc.).

system of record

the definitive and singular source of operational data. If data element abc has a value of 25 in a database record but a value of 45 in the system of record, by definition the first value is incorrect and must be reconciled. The system of record is useful for managing redundancy of data.

T-W

table

a relation that consists of a set of columns with a heading and a set of rows (i.e., tuples).

time stamping

the practice of tagging each record with some moment in time, usually when the record was created or when the record was passed from one environment to another.

time variant data

data whose accuracy is relevant to some moment in time. The three forms of time variant data are continuous time span, event discrete, and periodic discrete data. See current value data.

transition data

data possessing both primitive and derived characteristics; usually very sensitive to the running of the business. Typical transition data are interest rates for a bank, policy rates for an insurance company, retail sale rates for a manufacturer/distributor, etc.

trend analysis

the process of looking at homogeneous data over a spectrum of time. See EIS.

true archival data

data at the lowest level in the atomic database, usually stored on bulk storage media.

update

to change, add, delete, or replace values in all or selected entries, groups, or attributes stored in a database.

user

a person or process issuing commands and messages to the information system.

waterfall development

the classical approach to development where all development activity of one type is completed before the next development phase begins. The classical SDLC or structured approach to development.

Web

the network of Internet users.

Web log

the place in the Web site where detailed clickstream data is accumulated.