CHAPTER 1

Introduction to Data Warehousing

Information assets are immensely valuable to any enterprise, and because of this, these assets must be properly stored and readily accessible when they are needed. However, the availability of too much data makes the extraction of the most important information difficult, if not impossible. View results from any Google search, and you'll see that the *data = information* equation is not always correct—that is, too much data is simply too much.

Data warehousing is a phenomenon that grew from the huge amount of electronic data stored in recent years and from the urgent need to use that data to accomplish goals that go beyond the routine tasks linked to daily processing. In a typical scenario, a large corporation has many branches, and senior managers need to quantify and evaluate how each branch contributes to the global business performance. The corporate database stores detailed data on the tasks performed by branches. To meet the managers' needs, tailor-made queries can be issued to retrieve the required data. In order for this process to work, database administrators must first formulate the desired query (typically an aggregate SQL query) after closely studying database catalogs. Then the query is processed. This can take a few hours because of the huge amount of data, the query complexity, and the concurrent effects of other regular workload queries on data. Finally, a report is generated and passed to senior managers in the form of a spreadsheet.

Many years ago, database designers realized that such an approach is hardly feasible, because it is very demanding in terms of time and resources, and it does not always achieve the desired results. Moreover, a mix of analytical queries with transactional routine queries inevitably slows down the system, and this does not meet the needs of users of either type of query. Today's advanced data warehousing processes separate *online analytical processing (OLAP)* from *online transactional processing (OLTP)* by creating a new information repository that integrates basic data from various sources, properly arranges data formats, and then makes data available for analysis and evaluation aimed at planning and decision-making processes (Lechtenbörger, 2001).

Let's review some fields of application for which data warehouse technologies are successfully used:

- **Trade** Sales and claims analyses, shipment and inventory control, customer care and public relations
- **Craftsmanship** Production cost control, supplier and order support
- **Financial services** Risk analysis and credit cards, fraud detection
- **Transport industry** Vehicle management
- **Telecommunication services** Call flow analysis and customer profile analysis
- **Health care service** Patient admission and discharge analysis and bookkeeping in accounts departments

The field of application of data warehouse systems is not only restricted to enterprises, but it also ranges from epidemiology to demography, from natural science to education. A property that is common to all fields is the need for storage and query tools to retrieve information summaries easily and quickly from the huge amount of data stored in databases or made available by the Internet. This kind of information allows us to study business phenomena, learn about meaningful correlations, and gain useful knowledge to support decision-making processes.

1.1 Decision Support Systems

Until the mid-1980s, enterprise databases stored only *operational data*—data created by business operations involved in daily management processes, such as purchase management, sales management, and invoicing. However, every enterprise must have quick, comprehensive access to the information required by decision-making processes. This strategic information is extracted mainly from the huge amount of operational data stored in enterprise databases by means of a progressive selection and aggregation process shown in Figure 1-1.

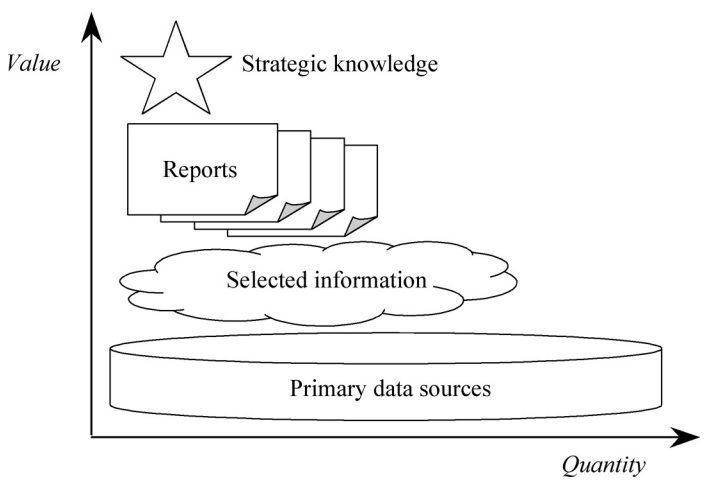


Figure 1-1 Information value as a function of quantity

An exponential increase in operational data has made computers the only tools suitable for providing data for decision-making performed by business managers. This fact has dramatically affected the role of enterprise databases and fostered the introduction of *decision support systems*. The concept of decision support systems mainly evolved from two research fields: theoretical studies on decision-making processes for organizations and technical research on interactive IT systems. However, the decision support system concept is based on several disciplines, such as databases, artificial intelligence, man-machine interaction, and simulation. Decision support systems became a research field in the mid-'70s and became more popular in the '80s.

Decision Support System
A *decision support system (DSS)* is a set of expandable, interactive IT techniques and tools designed for processing and analyzing data and for supporting managers in decision making. To do this, the system matches individual resources of managers with computer resources to improve the quality of the decisions made.

In practice, a DSS is an IT system that helps managers make decisions or choose among different alternatives. The system provides value estimates for each alternative, allowing the manager to critically review the results. Table 1-1 shows a possible classification of DSSs on the basis of their functions (Power, 2002).

From the architectural viewpoint, a DSS typically includes a model-based management system connected to a knowledge engine and, of course, an interactive graphical user interface (Sprague and Carlson, 1982). Data warehouse systems have been managing the data back-ends of DSSs since the 1990s. They must retrieve useful information from a huge amount of data stored on heterogeneous platforms. In this way, decision-makers can formulate their queries and conduct complex analyses on relevant information without slowing down operational systems.

System	Description
Passive DSS	Supports decision-making processes, but it does not offer explicit suggestions on decisions or solutions.
Active DSS	Offers suggestions and solutions.
Collaborative DSS	Operates interactively and allows decision-makers to modify, integrate, or refine suggestions given by the system. Suggestions are sent back to the system for validation.
Model-driven DSS	Enhances management of statistical, financial, optimization, and simulation models.
Communication-driven DSS	Supports a group of people working on a common task.
Data-driven DSS	Enhances the access and management of time series of corporate and external data.
Document-driven DSS	Manages and processes nonstructured data in many formats.
Knowledge-driven DSS	Provides problem-solving features in the form of facts, rules, and procedures.

Table 1-1 Classification of Decision Support Systems

1.2 Data Warehousing

Data warehouse systems are probably the systems to which academic communities and industrial bodies have been paying the greatest attention among all the DSSs. Data warehousing can be informally defined as follows:

Data Warehousing
Data warehousing is a collection of methods, techniques, and tools used to support *knowledge workers*—senior managers, directors, managers, and analysts—to conduct data analyses that help with performing decision-making processes and improving information resources.

The definition of data warehousing presented here is intentionally generic; it gives you an idea of the process but does not include specific features of the process. To understand the role and the useful properties of data warehousing completely, you must first understand the needs that brought it into being. In 1996, R. Kimball efficiently summed up a few claims frequently submitted by end users of classic information systems:

- "*We have heaps of data, but we cannot access it!*" This shows the frustration of those who are responsible for the future of their enterprises but have no technical tools to help them extract the required information in a proper format.
- "*How can people playing the same role achieve substantially different results?*" In midsize to large enterprises, many databases are usually available, each devoted to a specific business area. They are often stored on different logical and physical media that are not conceptually integrated. For this reason, the results achieved in every business area are likely to be inconsistent.
- "*We want to select, group, and manipulate data in every possible way!*" Decision-making processes cannot always be planned before the decisions are made. End users need a tool that is user-friendly and flexible enough to conduct ad hoc analyses. They want to choose which new correlations they need to search for in real time as they analyze the information retrieved.
- "*Show me just what matters!*" Examining data at the maximum level of detail is not only useless for decision-making processes, but is also self-defeating, because it does not allow users to focus their attention on meaningful information.
- "*Everyone knows that some data is wrong!*" This is another sore point. An appreciable percentage of transactional data is not correct—or it is unavailable. It is clear that you cannot achieve good results if you base your analyses on incorrect or incomplete data.

We can use the previous list of problems and difficulties to extract a list of key words that become distinguishing marks and essential requirements for a *data warehouse process*, a set of tasks that allow us to turn operational data into decision-making support information:

- *accessibility* to users not very familiar with IT and data structures;
- *integration* of data on the basis of a standard enterprise model;
- *query flexibility* to maximize the advantages obtained from the existing information;

- *information conciseness* allowing for target-oriented and effective analyses;
- *multidimensional representation* giving users an intuitive and manageable view of information;
- *correctness and completeness* of integrated data.

Data warehouses are placed right in the middle of this process and act as repositories for data. They make sure that the requirements set can be fulfilled.

Data Warehouse

A *data warehouse* is a collection of data that supports decision-making processes. It provides the following features (Inmon, 2005):

- It is subject-oriented.
- It is integrated and consistent.
- It shows its evolution over time and it is not volatile.

Data warehouses are subject-oriented because they hinge on enterprise-specific concepts, such as customers, products, sales, and orders. On the contrary, operational databases hinge on many different enterprise-specific applications.

We put emphasis on integration and consistency because data warehouses take advantage of multiple data sources, such as data extracted from production and then stored to enterprise databases, or even data from a third party's information systems. A data warehouse should provide a unified view of all the data. Generally speaking, we can state that creating a data warehouse system does not require that new information be added; rather, existing information needs rearranging. This implicitly means that an information system should be previously available.

Operational data usually covers a short period of time, because most transactions involve the latest data. A data warehouse should enable analyses that instead cover a few years. For this reason, data warehouses are regularly updated from operational data and keep on growing. If data were visually represented, it might progress like so: A photograph of operational data would be made at regular intervals. The sequence of photographs would be stored to a data warehouse, and results would be shown in a movie that reveals the status of an enterprise from its foundation until present.

Fundamentally, data is never deleted from data warehouses and updates are normally carried out when data warehouses are offline. This means that data warehouses can be essentially viewed as read-only databases. This satisfies the users' need for a short analysis query response time and has other important effects. First, it affects data warehouse–specific database management system (DBMS) technologies, because there is no need for advanced transaction management techniques required by operational applications. Second, data warehouses operate in read-only mode, so data warehouse–specific logical design solutions are completely different from those used for operational databases. For instance, the most obvious feature of data warehouse relational implementations is that table normalization can be given up to partially denormalize tables and improve performance.

Other differences between operational databases and data warehouses are connected with query types. Operational queries execute transactions that generally read/write a

small number of tuples from/to many tables connected by simple relations. For example, this applies if you search for the data of a customer in order to insert a new customer order. This kind of query is an OLTP query. On the contrary, the type of query required in data warehouses is OLAP. It features dynamic, multidimensional analyses that need to scan a huge amount of records to process a set of numeric data summing up the performance of an enterprise. It is important to note that OLTP systems have an essential workload core "frozen" in application programs, and ad hoc data queries are occasionally run for data maintenance. Conversely, data warehouse interactivity is an essential property for analysis sessions, so the actual workload constantly changes as time goes by.

The distinctive features of OLAP queries suggest adoption of a *multidimensional* representation for data warehouse data. Basically, data is viewed as points in space, whose dimensions correspond to many possible analysis dimensions. Each point represents an event that occurs in an enterprise and is described by a set of measures relevant to decision-making processes. Section 1.5 gives a detailed description of the multidimensional model you absolutely need to be familiar with to understand how to model conceptual and logical levels of a data warehouse and how to query data warehouses.

Table 1-2 summarizes the main differences between operational databases and data warehouses.

NOTE *For further details on the different issues related to the data warehouse process, refer to Chaudhuri and Dayal, 1997; Inmon, 2005; Jarke et al., 2000; Kelly, 1997; Kimball, 1996; Mattison, 2006; and Wrembel and Koncilia, 2007.*

Feature	Operational Databases	Data Warehouses
Users	Thousands	Hundreds
Workload	Preset transactions	Specific analysis queries
Access	To hundreds of records, write and read mode	To millions of records, mainly read-only mode
Goal	Depends on applications	Decision-making support
Data	Detailed, both numeric and alphanumeric	Summed up, mainly numeric
Data integration	Application-based	Subject-based
Quality	In terms of integrity	In terms of consistency
Time coverage	Current data only	Current and historical data
Updates	Continuous	Periodical
Model	Normalized	Denormalized, multidimensional
Optimization	For OLTP access to a database part	For OLAP access to most of the database

TABLE 1-2 Differences Between Operational Databases and Data Warehouses (Kelly, 1997)

1.3 Data Warehouse Architectures

The following architecture properties are essential for a data warehouse system (Kelly, 1997):

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

Two different classifications are commonly adopted for data warehouse architectures. The first classification, described in sections 1.3.1, 1.3.2, and 1.3.3, is a structure-oriented one that depends on the number of layers used by the architecture. The second classification, described in section 1.3.4, depends on how the different layers are employed to create enterprise-oriented or department-oriented views of data warehouses.

1.3.1 Single-Layer Architecture

A single-layer architecture is not frequently used in practice. Its goal is to minimize the amount of data stored; to reach this goal, it removes data redundancies. Figure 1-2 shows the only layer physically available: the source layer. In this case, data warehouses are *virtual*.

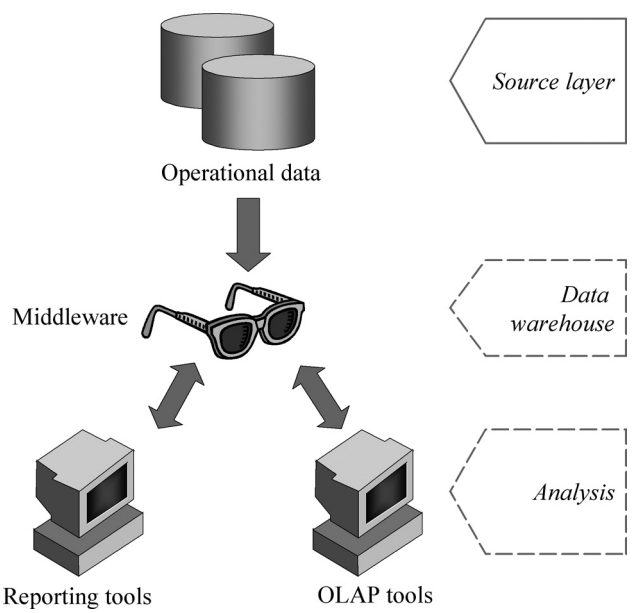


Figure 1-2 Single-layer architecture for a data warehouse system

This means that a data warehouse is implemented as a multidimensional view of operational data created by specific *middleware*, or an intermediate processing layer (Devlin, 1997).

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

1.3.2 Two-Layer Architecture

The requirement for separation plays a fundamental role in defining the typical architecture for a data warehouse system, as shown in Figure 1-3. Although it is typically called a *two-layer architecture* to highlight a separation between physically available sources and data warehouses, it actually consists of four subsequent data flow stages (Lechtenbörger, 2001):

1. **Source layer** A data warehouse system uses heterogeneous sources of data. That data is originally stored to corporate relational databases or *legacy*[1] databases, or it may come from information systems outside the corporate walls.
2. **Data staging** The data stored to sources should be extracted, cleansed to remove inconsistencies and fill gaps, and integrated to merge heterogeneous sources into one common schema. The so-called *Extraction, Transformation, and Loading tools (ETL)* can merge heterogeneous schemata, extract, transform, cleanse, validate, filter, and load source data into a data warehouse (Jarke et al., 2000). Technologically speaking, this stage deals with problems that are typical for distributed information systems, such as inconsistent data management and incompatible data structures (Zhuge et al., 1996). Section 1.4 deals with a few points that are relevant to data staging.
3. **Data warehouse layer** Information is stored to one logically centralized single repository: a data warehouse. The data warehouse can be directly accessed, but it can also be used as a source for creating *data marts*, which partially replicate data warehouse contents and are designed for specific enterprise departments. *Meta-data repositories* (section 1.6) store information on sources, access procedures, data staging, users, data mart schemata, and so on.
4. **Analysis** In this layer, integrated data is efficiently and flexibly accessed to issue reports, dynamically analyze information, and simulate hypothetical business scenarios. Technologically speaking, it should feature aggregate data navigators, complex query optimizers, and user-friendly GUIs. Section 1.7 deals with different types of decision-making support analyses.

The architectural difference between *data warehouses* and *data marts* needs to be studied closer. The component marked as a data warehouse in Figure 1-3 is also often called the *primary data warehouse* or *corporate data warehouse*. It acts as a centralized storage system for

[1]The term *legacy system* denotes corporate applications, typically running on mainframes or minicomputers, that are currently used for operational tasks but do not meet modern architectural principles and current standards. For this reason, accessing legacy systems and integrating them with more recent applications is a complex task. All applications that use a nonrelational database are examples of legacy systems.

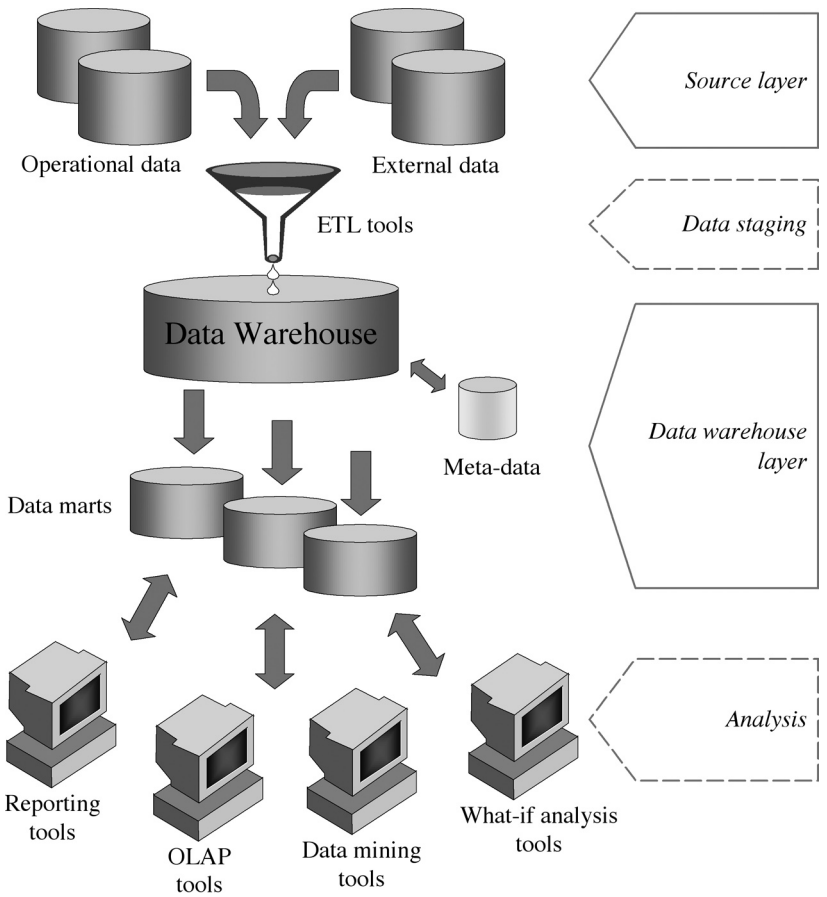


FIGURE 1-3
Two-layer architecture for a data warehouse system

all the data being summed up. Data marts can be viewed as small, local data warehouses replicating (and summing up as much as possible) the part of a primary data warehouse required for a specific application domain.

Data Marts
A *data mart* is a subset or an aggregation of the data stored to a primary data warehouse. It includes a set of information pieces relevant to a specific business area, corporate department, or category of users.

The data marts populated from a primary data warehouse are often called *dependent*. Although data marts are not strictly necessary, they are very useful for data warehouse systems in midsize to large enterprises because

- they are used as building blocks while incrementally developing data warehouses;
- they mark out the information required by a specific group of users to solve queries;
- they can deliver better performance because they are smaller than primary data warehouses.

Sometimes, mainly for organization and policy purposes, you should use a different architecture in which sources are used to directly populate data marts. These data marts are called *independent* (see section 1.3.4). If there is no primary data warehouse, this streamlines the design process, but it leads to the risk of inconsistencies between data marts. To avoid these problems, you can create a primary data warehouse and still have independent data marts. In comparison with the standard two-layer architecture of Figure 1-3, the roles of data marts and data warehouses are actually inverted. In this case, the data warehouse is populated from its data marts, and it can be directly queried to make access patterns as easy as possible.

The following list sums up all the benefits of a two-layer architecture, in which a data warehouse separates sources from analysis applications (Jarke et al., 2000; Lechtenbörger, 2001):

- In data warehouse systems, good quality information is always available, even when access to sources is denied temporarily for technical or organizational reasons.
- Data warehouse analysis queries do not affect the management of transactions, the reliability of which is vital for enterprises to work properly at an operational level.
- Data warehouses are logically structured according to the multidimensional model, while operational sources are generally based on relational or semi-structured models.
- A mismatch in terms of time and granularity occurs between OLTP systems, which manage current data at a maximum level of detail, and OLAP systems, which manage historical and summarized data.
- Data warehouses can use specific design solutions aimed at performance optimization of analysis and report applications.

NOTE *A few authors use the same terminology to define different concepts. In particular, those authors consider a data warehouse as a repository of integrated and consistent, yet operational, data, while they use a multidimensional representation of data only in data marts. According to our terminology, this "operational view" of data warehouses essentially corresponds to the reconciled data layer in three-layer architectures.*

1.3.3 Three-Layer Architecture

In this architecture, the third layer is the *reconciled data layer* or *operational data store*. This layer materializes operational data obtained after integrating and cleansing source data. As a result, those data are integrated, consistent, correct, current, and detailed. Figure 1-4 shows a data warehouse that is not populated from its sources directly, but from reconciled data.

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

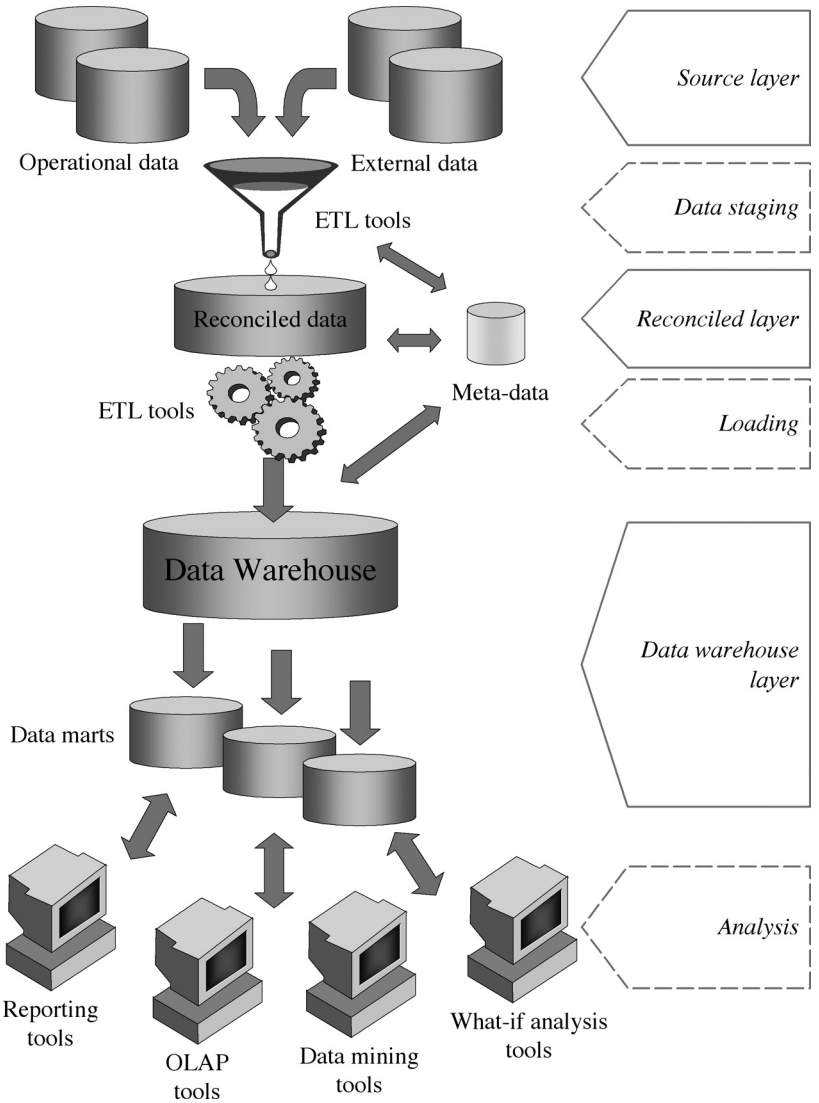


FIGURE 1-4 Three-layer architecture for a data warehouse system

Finally, let's consider a supplementary architectural approach, which provides a comprehensive picture. This approach can be described as a hybrid solution between the single-layer architecture and the two/three-layer architecture. This approach assumes that although a data warehouse is available, it is unable to solve all the queries formulated. This means that users may be interested in directly accessing source data from aggregate data (*drill-through*). To reach this goal, some queries have to be rewritten on the basis of source data (or reconciled data if it is available). This type of architecture is implemented in a prototype by Cui and Widom, 2000, and it needs to be able to go dynamically back to the source data required for queries to be solved (*lineage*).

NOTE *Gupta, 1997a; Hull and Zhou, 1996; and Yang et al., 1997 discuss the implications of this approach from the viewpoint of performance optimization, and in particular view materialization.*

1.3.4 An Additional Architecture Classification

The scientific literature often distinguishes five types of architecture for data warehouse systems, in which the same basic layers mentioned in the preceding paragraphs are combined in different ways (Rizzi, 2008).

In *independent data marts architecture*, different data marts are separately designed and built in a nonintegrated fashion (Figure 1-5). This architecture can be initially adopted in the absence of a strong sponsorship toward an enterprise-wide warehousing project, or when the organizational divisions that make up the company are loosely coupled. However, it tends to be soon replaced by other architectures that better achieve data integration and cross-reporting.

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

In the *hub-and-spoke architecture*, one of the most used in medium to large contexts, there is much attention to scalability and extensibility, and to achieving an enterprise-wide view

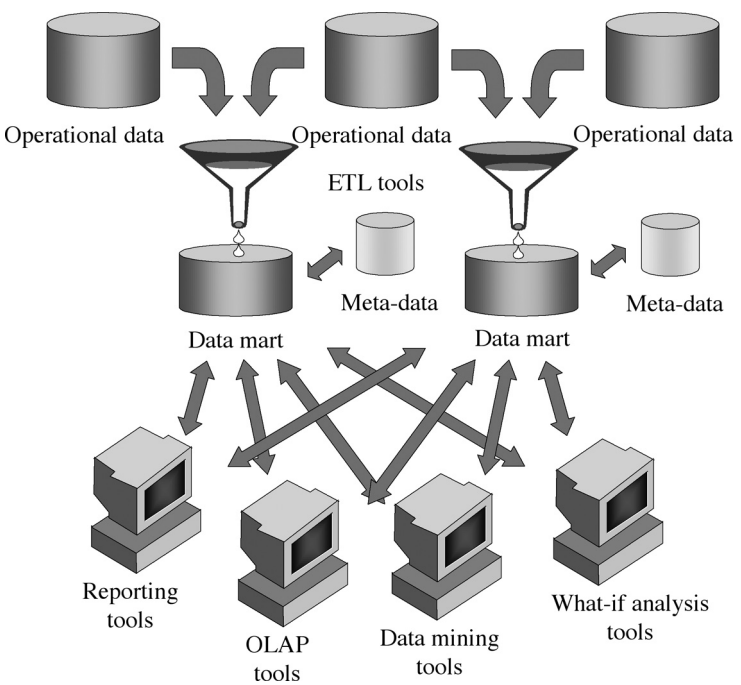


FIGURE 1-5 Independent data marts architecture

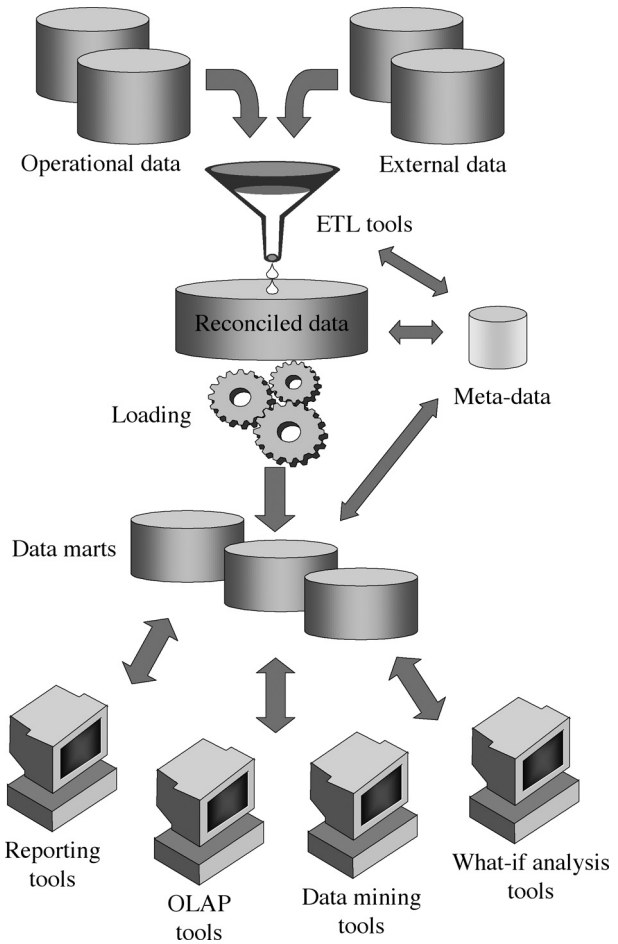


FIGURE 1-6
Hub-and-spoke architecture

of information. Atomic, normalized data is stored in a reconciled layer that feeds a set of data marts containing summarized data in multidimensional form (Figure 1-6). Users mainly access the data marts, but they may occasionally query the reconciled layer.

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

The *federated architecture* is sometimes adopted in dynamic contexts where preexisting data warehouses/data marts are to be noninvasively integrated to provide a single, cross-organization decision support environment (for instance, in the case of mergers and acquisitions). Each data warehouse/data mart is either virtually or physically integrated with the others, leaning on a variety of advanced techniques such as distributed querying, ontologies, and meta-data interoperability (Figure 1-7).

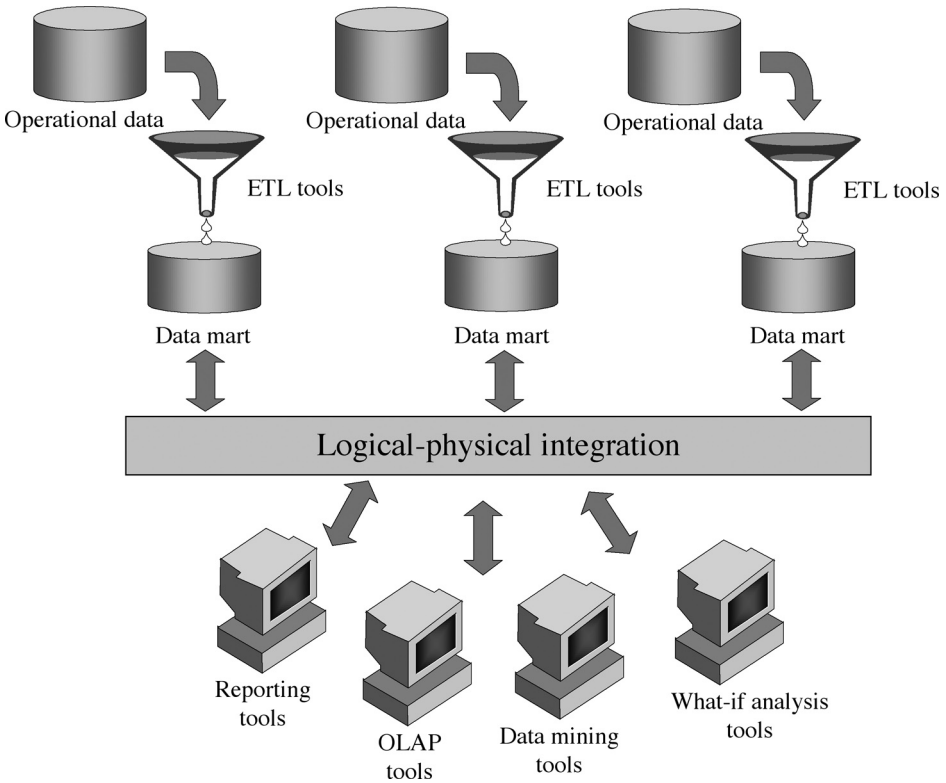


Figure 1-7 Federated architecture

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

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

1.4 Data Staging and ETL

Now let's closely study some basic features of the different architecture layers. We will start with the data staging layer.

The data staging layer hosts the ETL processes that extract, integrate, and clean data from operational sources to feed the data warehouse layer. In a three-layer architecture, ETL processes actually feed the reconciled data layer—a single, detailed, comprehensive, top-quality data source—that in its turn feeds the data warehouse. For this reason, the ETL process operations as a whole are often defined as *reconciliation*. These are also the most complex and technically challenging among all the data warehouse process phases.

ETL takes place once when a data warehouse is populated for the first time, then it occurs every time the data warehouse is regularly updated. Figure 1-8 shows that ETL consists of four separate phases: *extraction* (or *capture*), *cleansing* (or *cleaning* or *scrubbing*), *transformation,* and *loading*. In the following sections, we offer brief descriptions of these phases.

NOTE *Refer to Jarke et al., 2000; Hoffer et al., 2005; Kimball and Caserta, 2004; and English, 1999 for more details on ETL.*

The scientific literature shows that the boundaries between cleansing and transforming are often blurred from the terminological viewpoint. For this reason, a specific operation is not always clearly assigned to one of these phases. This is obviously a formal problem, but not a substantial one. We will adopt the approach used by Hoffer and others (2005) to make our explanations as clear as possible. Their approach states that cleansing is essentially aimed at rectifying data *values*, and transformation more specifically manages data *formats*.

Chapter 10 discusses all the details of the data-staging design phase. Chapter 3 deals with an early data warehouse design phase: *integration*. This phase is necessary if there are heterogeneous sources to define a schema for the reconciled data layer, and to specifically transform operational data in the data-staging phase.

1.4.1 Extraction

Relevant data is obtained from sources in the extraction phase. You can use *static extraction* when a data warehouse needs populating for the first time. Conceptually speaking, this looks like a snapshot of operational data. *Incremental extraction*, used to update data warehouses regularly, seizes the changes applied to source data since the latest extraction. Incremental extraction is often based on the log maintained by the operational DBMS. If a timestamp is associated with operational data to record exactly when the data is changed or added, it can be used to streamline the extraction process. Extraction can also be source-driven if you can rewrite operational applications to asynchronously notify of the changes being applied, or if your operational database can implement triggers associated with change transactions for relevant data.

The data to be extracted is mainly selected on the basis of its quality (English, 1999). In particular, this depends on how comprehensive and accurate the constraints implemented in sources are, how suitable the data formats are, and how clear the schemata are.

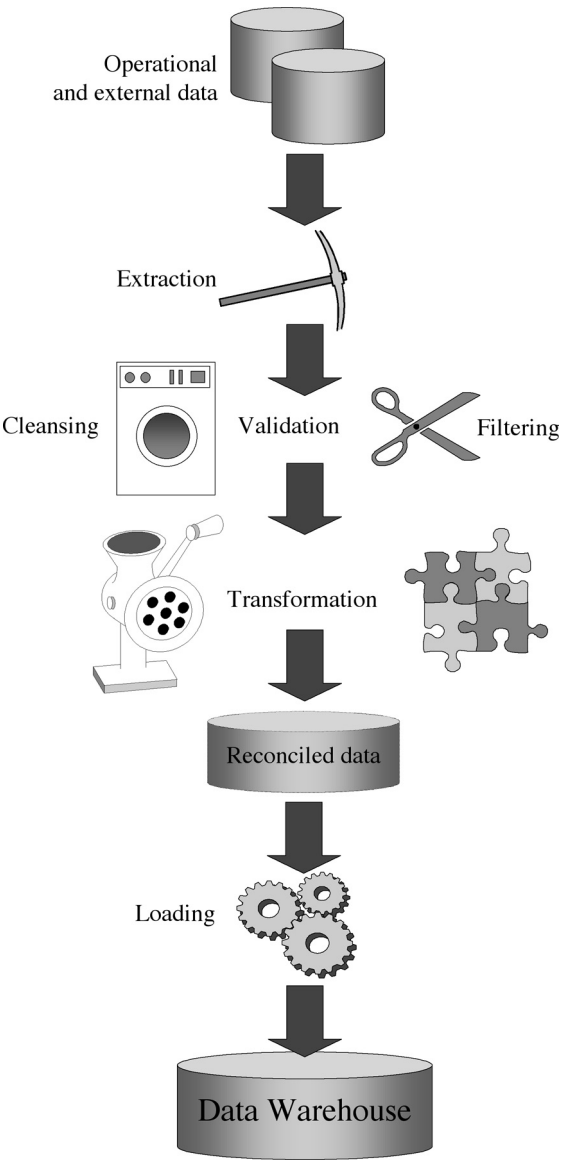


Figure 1-8 Extraction, transformation, and loading

1.4.2 Cleansing

The cleansing phase is crucial in a data warehouse system because it is supposed to improve data quality—normally quite poor in sources (Galhardas et al., 2001). The following list includes the most frequent mistakes and inconsistencies that make data "dirty":

- **Duplicate data** For example, a patient is recorded many times in a hospital patient management system

- **Inconsistent values that are logically associated** Such as addresses and ZIP codes
- **Missing data** Such as a customer's job
- **Unexpected use of fields** For example, a `socialSecurityNumber` field could be used improperly to store office phone numbers
- **Impossible or wrong values** Such as 2/30/2009
- **Inconsistent values for a single entity because different practices were used** For example, to specify a country, you can use an international country abbreviation (I) or a full country name (Italy); similar problems arise with addresses (Hamlet Rd. and Hamlet Road)
- **Inconsistent values for one individual entity because of typing mistakes** Such as Hamet Road instead of Hamlet Road

In particular, note that the last two types of mistakes are very frequent when you are managing multiple sources and are entering data manually.

The main data cleansing features found in ETL tools are *rectification* and *homogenization*. They use specific dictionaries to rectify typing mistakes and to recognize synonyms, as well as *rule-based cleansing* to enforce domain-specific rules and define appropriate associations between values. See section 10.2 for more details on these points.

1.4.3 Transformation

Transformation is the core of the reconciliation phase. It converts data from its operational source format into a specific data warehouse format. If you implement a three-layer architecture, this phase outputs your reconciled data layer. Independently of the presence of a reconciled data layer, establishing a mapping between the source data layer and the data warehouse layer is generally made difficult by the presence of many different, heterogeneous sources. If this is the case, a complex integration phase is required when designing your data warehouse. See Chapter 3 for more details.

The following points must be rectified in this phase:

- *Loose texts may hide valuable information.* For example, BigDeal LtD does not explicitly show that this is a Limited Partnership company.
- *Different formats can be used for individual data.* For example, a date can be saved as a string or as three integers.

Following are the main transformation processes aimed at populating the reconciled data layer:

- *Conversion* and *normalization* that operate on both storage formats and units of measure to make data uniform
- *Matching* that associates equivalent fields in different sources
- *Selection* that reduces the number of source fields and records

When populating a data warehouse, normalization is replaced by *denormalization* because data warehouse data are typically denormalized, and you need *aggregation* to sum up data properly.

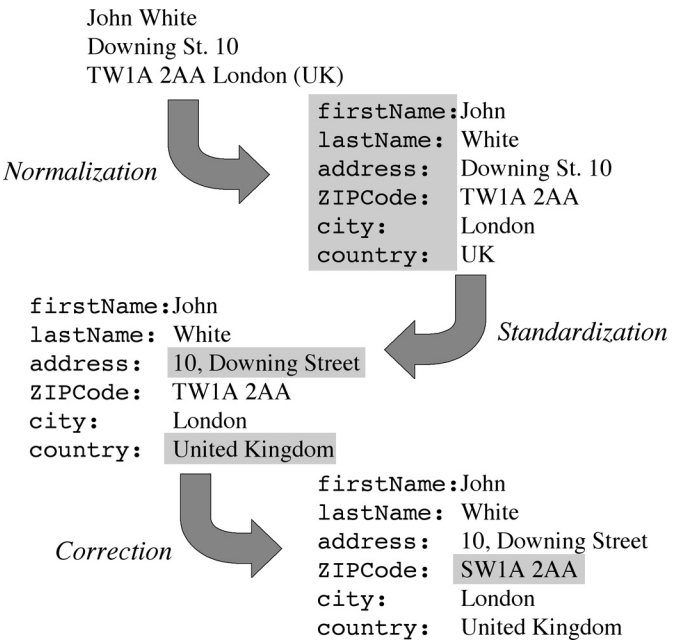


Figure 1-9 Example of cleansing and transforming customer data

Cleansing and transformation processes are often closely connected in ETL tools. Figure 1-9 shows an example of cleansing and transformation of customer data: a field-based structure is extracted from a loose text, then a few values are standardized so as to remove abbreviations, and eventually those values that are logically associated can be rectified.

1.4.4 Loading

Loading into a data warehouse is the last step to take. Loading can be carried out in two ways:

- **Refresh** Data warehouse data is completely rewritten. This means that older data is replaced. Refresh is normally used in combination with static extraction to initially populate a data warehouse.
- **Update** Only those changes applied to source data are added to the data warehouse. Update is typically carried out without deleting or modifying preexisting data. This technique is used in combination with incremental extraction to update data warehouses regularly.

1.5 Multidimensional Model

The data warehouse layer is a vitally important part of this book. Here, we introduce a data warehouse key word: *multidimensional*. You need to become familiar with the concepts and terminology used here to understand the information presented throughout this book, particularly information regarding conceptual and logical modeling and designing.

Over the last few years, multidimensional databases have generated much research and market interest because they are fundamental for many decision-making support applications, such as data warehouse systems. The reason why the multidimensional model is used as a paradigm of data warehouse data representation is fundamentally connected to its ease of use and intuitiveness even for IT newbies. The multidimensional model's success is also linked to the widespread use of productivity tools, such as spreadsheets, that adopt the multidimensional model as a visualization paradigm.

Perhaps the best starting point to approach the multidimensional model effectively is a definition of the types of queries for which this model is best suited. Section 1.7 offers more details on typical decision-making queries such as those listed here (Jarke et al., 2000):

- "*What is the total amount of receipts recorded last year per state and per product category?*"
- "*What is the relationship between the trend of PC manufacturers' shares and quarter gains over the last five years?*"
- "*Which orders maximize receipts?*"
- "*Which one of two new treatments will result in a decrease in the average period of admission?*"
- "*What is the relationship between profit gained by the shipments consisting of less than 10 items and the profit gained by the shipments of more than 10 items?*"

It is clear that using traditional languages, such as SQL, to express these types of queries can be a very difficult task for inexperienced users. It is also clear that running these types of queries against operational databases would result in an unacceptably long response time.

The multidimensional model begins with the observation that the factors affecting decision-making processes are enterprise-specific *facts*, such as sales, shipments, hospital admissions, surgeries, and so on. Instances of a fact correspond to *events* that occurred. For example, every single sale or shipment carried out is an event. Each fact is described by the values of a set of relevant *measures* that provide a quantitative description of events. For example, sales receipts, amounts shipped, hospital admission costs, and surgery time are measures.

Obviously, a huge number of events occur in typical enterprises—too many to analyze one by one. Imagine placing them all into an *n*-dimensional space to help us quickly select and sort them out. The *n*-dimensional space axes are called *analysis dimensions,* and they define different perspectives to single out events. For example, the sales in a store chain can be represented in a three-dimensional space whose dimensions are products, stores, and dates. As far as shipments are concerned, products, shipment dates, orders, destinations, and terms & conditions can be used as dimensions. Hospital admissions can be defined by the department-date-patient combination, and you would need to add the type of operation to classify surgery operations.

The concept of dimension gave life to the broadly used metaphor of *cube*s to represent multidimensional data. According to this metaphor, events are associated with cube cells and cube edges stand for analysis dimensions. If more than three dimensions exist, the cube is called a *hypercube*. Each cube cell is given a value for each measure. Figure 1-10 shows an intuitive representation of a cube in which the fact is a sale in a store chain. Its analysis dimensions are *store*, *product* and *date*. An event stands for a specific item sold in a specific store on a specific date, and it is described by two measures: the *quantity* sold and the *receipts*. This figure highlights that the cube is *sparse*—this means that many events did not actually take place. Of course, you cannot sell every item every day in every store.

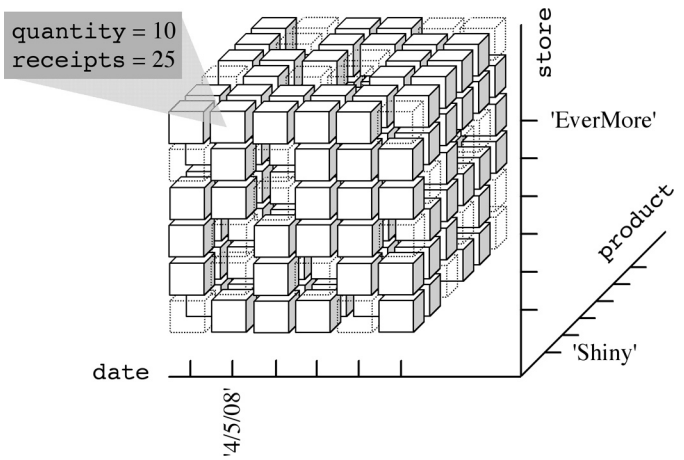


FIGURE 1-10 The three-dimensional cube modeling sales in a store chain: 10 packs of Shiny were sold on 4/5/2008 in the EverMore store, totaling $25.

If you want to use the relational model to represent this cube, you could use the following relational schema:

```
SALES(store, product, date, quantity, receipts)
```

Here, the underlined attributes make up the primary key and events are associated with tuples, such as <'EverMore', 'Shiny', '04/05/08', 10, 25>. The constraint expressed by this primary key specifies that two events cannot be associated with an individual store, product, and date value combination, and that every value combination *functionally determines* a unique value for `quantity` and a unique value for `receipts`. This means that the following functional dependency[2] holds:

```
store, product, date → quantity, receipts
```

To avoid any misunderstanding of the term *event*, you should realize that the group of dimensions selected for a fact representation singles out a unique event in the multidimensional model, but the group does not necessarily single out a unique event in the application domain. To make this statement clearer, consider once again the sales example. In the application domain, one single *sales* event is supposed to be a customer's purchase of a set of products from a store on a specific date. In practice, this corresponds to a sales receipt. From the viewpoint of the multidimensional model, if the sales fact has the `product`, `store`, and `date` dimensions, an event will be the daily total amount of an item sold in a store. It is clear that the difference between both interpretations depends on sales

[2]The definition of functional dependency belongs to relational theory. Given relation schema R and two attribute sets $X=\{a_1 \ldots, a_n\}$ and $Y=\{b_1 \ldots, b_m\}$, X is said to *functionally determine* Y ($X \rightarrow Y$) if and only if, for every legal instance r of R and for each pair of tuples t_1, t_2 in r, $t_1[X]=t_2[X]$ implies $t_1[Y]=t_2[Y]$. Here $t[X/Y]$ denotes the values taken in t from the attributes in X/Y. By extension, we say that a functional dependency holds between two attribute sets X and Y when each value set of X always corresponds to a single value set of Y. To simplify the notation, when we denote the attributes in each set, we drop the braces.

receipts that generally include various items, and on individual items that are generally sold many times every day in a store. In the following sections, we use the terms *event* and *fact* to make reference to the granularity taken by events and facts in the multidimensional model.

Normally, each dimension is associated with a *hierarchy* of aggregation levels, often called *roll-up hierarchy*. Roll-up hierarchies group aggregation level values in different ways. Hierarchies consist of levels called *dimensional attributes*. Figure 1-11 shows a simple example of hierarchies built on the `product` and `store` dimensions: products are classified into types, and are then further classified into categories. Stores are located in cities belonging to states. On top of each hierarchy is a fake level that includes all the dimension-related values. From the viewpoint of relational theory, you can use a set of functional dependencies between dimensional attributes to express a hierarchy:

```
product → type → category
   store → city → state
```

In summary, a *multidimensional* cube hinges on a *fact* relevant to decision-making. It shows a set of *events* for which numeric *measures* provide a quantitative description. Each cube axis shows a possible analysis *dimension*. Each dimension can be analyzed at different detail levels specified by hierarchically structured *attributes*.

The scientific literature shows many formal expressions of the multidimensional model, which can be more or less complex and comprehensive. We'll briefly mention alternative terms used for the multidimensional model in the scientific literature and in commercial tools.

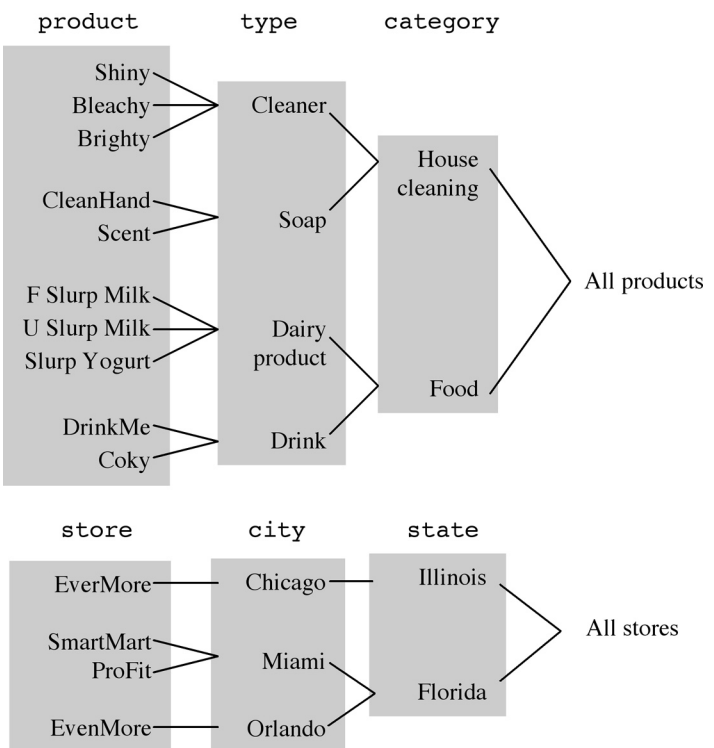


Figure 1-11 Aggregation hierarchies built on the `product` and `store` dimensions

The *fact* and *cube* terms are often interchangeably used. Essentially, everyone agrees on the use of the term *dimensions* to specify the coordinates that classify and identify fact occurrences. However, entire hierarchies are sometimes called *dimensions*. For example, the term *time dimension* can be used for the entire hierarchy built on the `date` attribute. Measures are sometimes called *variables, metrics, properties, attributes,* or *indicators.* In some models, dimensional attributes of hierarchies are called *levels* or *parameters.*

NOTE *The main formal expressions of the multidimensional model in the literature were proposed by Agrawal et al., 1995; Gyssens and Lakshmanan, 1997; Datta and Thomas, 1997; Vassiliadis, 1998; and Cabibbo and Torlone, 1998.*

The information in a multidimensional cube is very difficult for users to manage because of its quantity, even if it is a concise version of the information stored to operational databases. If, for example, a store chain includes 50 stores selling 1000 items, and a specific data warehouse covers three-year-long transactions (approximately 1000 days), the number of potential events totals $50 \times 1000 \times 1000 = 5 \times 10^7$. Assuming that each store can sell only 10 percent of all the available items per day, the number of events totals 5×10^6. This is still too much data to be analyzed by users without relying on automatic tools.

You have essentially two ways to reduce the quantity of data and obtain useful information: *restriction* and *aggregation*. The cube metaphor offers an easy-to-use and intuitive way to understand both of these methods, as we will discuss in the following paragraphs.

1.5.1 Restriction

Restricting data means separating part of the data from a cube to mark out an analysis field. In relational algebra terminology, this is called making *selections* and/or *projections*.

The simplest type of selection is *data slicing*, shown in Figure 1-12. When you slice data, you decrease cube dimensionality by setting one or more dimensions to a specific value. For example, if you set one of the sales cube dimensions to a value, such as `store='EverMore'`, this results in the set of events associated with the items sold in the EverMore store. According to the cube metaphor, this is simply a plane of cells—that is, a data slice that can be easily displayed in spreadsheets. In the store chain example given earlier, approximately 10^5 events still appear in your result. If you set two dimensions to a value, such as `store='EverMore'` and `date='4/5/2008'`, this will result in all the different items sold in the EverMore store on April 5 (approximately 100 events). Graphically speaking, this information is stored at the intersection of two perpendicular planes resulting in a line. If you set all the dimensions to a particular value, you will define just one event that corresponds to a point in the three-dimensional space of sales.

Dicing is a generalization of slicing. It poses some constraints on dimensional attributes to scale down the size of a cube. For example, you can select only the daily sales of the food items in April 2008 in Florida (Figure 1-12). In this way, if five stores are located in Florida and 50 food products are sold, the number of events to examine changes to $5 \times 50 \times 30 = 7500$.

Finally, a *projection* can be referred to as a choice to keep just one subgroup of measures for every event and reject other measures.

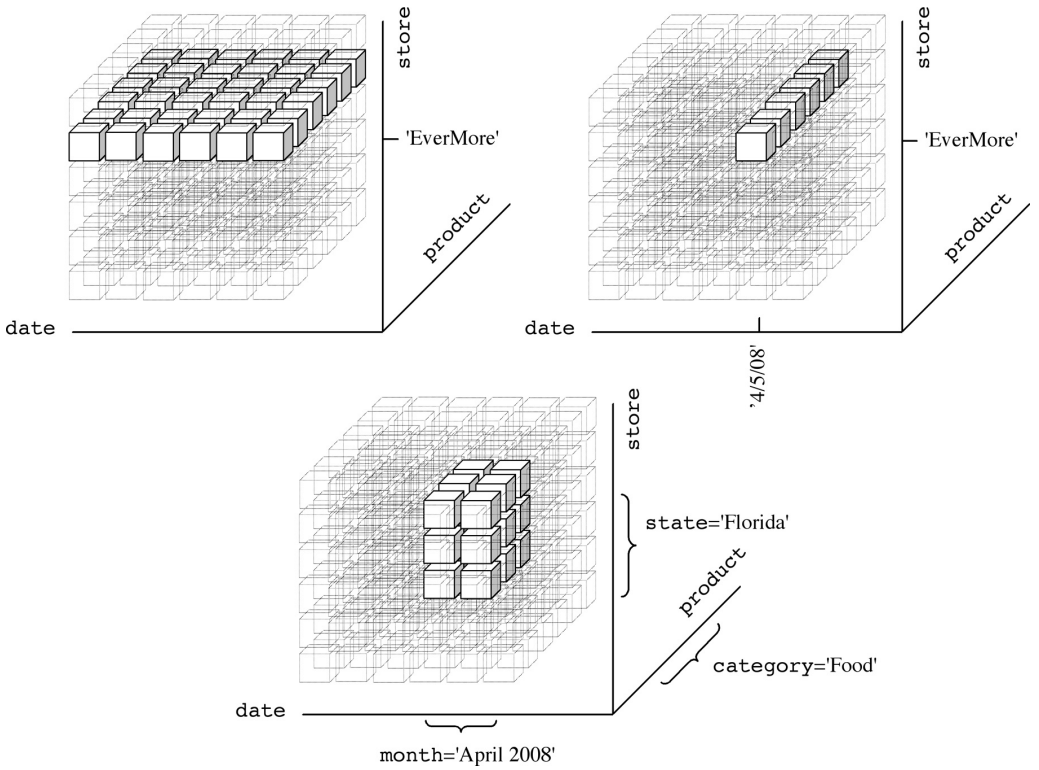


FIGURE 1-12 Slicing and dicing a three-dimensional cube

1.5.2 Aggregation

Aggregation plays a fundamental role in multidimensional databases. Assume, for example, that you want to analyze the items sold monthly for a three year period. According to the cube metaphor, this means that you need to sort all the cells related to the days of each month by product and store, and then merge them into one single macrocell. In the aggregate cube obtained in this way, the total number of events (that is, the number of macrocells) is $50 \times 1000 \times 36$. This is because the granularity of the time dimensions does not depend on days any longer, but now depends on months, and 36 is the number of months in three years. Every aggregate event will then sum up the data available in the events it aggregates. In this example, the total amount of items sold per month and the total receipts are calculated by summing every single value of their measures (Figure 1-13). If you further aggregate along time, you can achieve just three events for every store-product combination: one for every year. When you completely aggregate along the time dimension, each store-product combination corresponds to one single event, which shows the total amount of items sold in a store over three years and the total amount of receipts.

	EverMore	EvenMore	SmartMart
1/1/2007	–	–	–
1/2/2007	10	15	5
1/3/2007	20	–	5
.....	.....	.....	.....
1/1/2008	–	–	–
1/2/2008	15	10	20
1/3/2008	20	20	25
.....	.....	.....	.....
1/1/2009	–	–	–
1/2/2009	20	8	25
1/3/2009	20	12	20
.....	.....	.....	.....

	EverMore	EvenMore	SmartMart
January 2007	200	180	150
February 2007	180	150	120
March 2007	220	180	160
.....	.....	.....	.....
January 2008	350	220	200
February 2008	300	200	250
March 2008	310	180	300
.....	.....	.....	.....
January 2009	380	200	220
February 2009	310	200	250
March 2009	300	160	280
.....	.....	.....	.....

	EverMore	EvenMore	SmartMart
2007	2,400	2,000	1,600
2008	3,200	2,300	3,000
2009	3,400	2,200	3,200

	EverMore	EvenMore	SmartMart
Total	9,000	6,500	7,800

FIGURE 1-13 Time hierarchy aggregation of the quantity of items sold per product in three stores. A dash shows that an event did not occur because no item was sold.

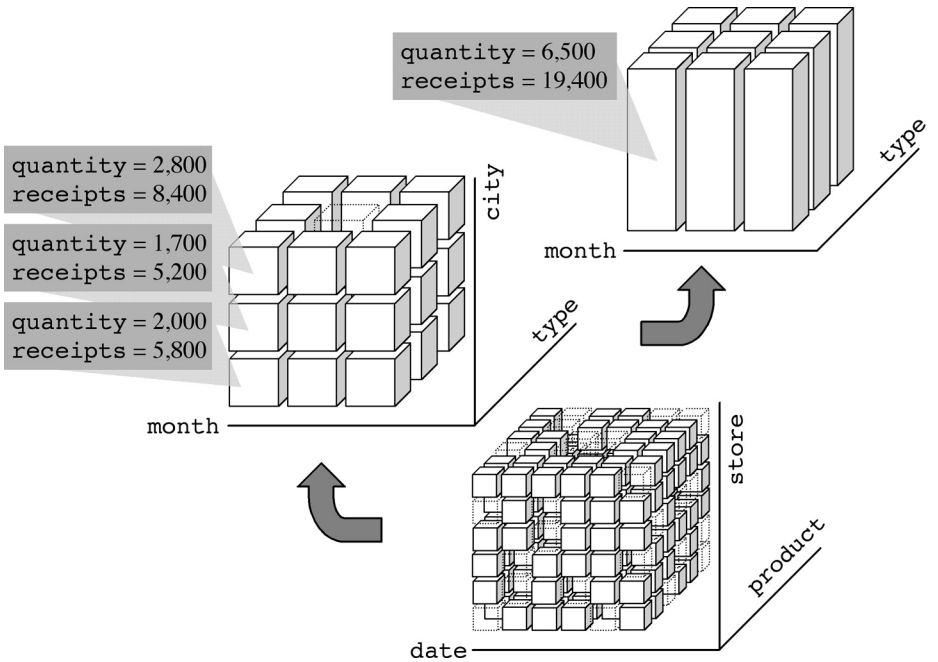


FIGURE 1-14 Two cube aggregation levels. Every macro-event measure value is a sum of its component event values.

You can aggregate along various dimensions at the same time. For example, Figure 1-14 shows that you can group sales by month, product type, and store city, and by month and product type. Moreover, selections and aggregations can be combined to carry out an analysis process targeted exactly to users' needs.

1.6 Meta-data

The term *meta-data* can be applied to the data used to define other data. In the scope of data warehousing, meta-data plays an essential role because it specifies source, values, usage, and features of data warehouse data and defines how data can be changed and processed at every architecture layer. Figures 1-3 and 1-4 show that the meta-data repository is closely connected to the data warehouse. Applications use it intensively to carry out data-staging and analysis tasks.

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

Meta-data is stored in a meta-data repository which all the other architecture components can access. According to Kelly, a tool for meta-data management should

- allow administrators to perform system administration operations, and in particular manage security;
- allow end users to navigate and query meta-data;
- use a GUI;
- allow end users to extend meta-data;
- allow meta-data to be imported/exported into/from other standard tools and formats.

As far as representation formats are concerned, Object Management Group (OMG, 2000) released a standard called *Common Warehouse Metamodel (CWM*) that relies on three famous standards: *Unified Modeling Language (UML)*, *eXtensible Markup Language (XML)*, and *XML Metadata Interchange (XMI)*. Partners, such as IBM, Unisys, NCR, and Oracle, in a common effort, created the new standard format that specifies how meta-data can be exchanged among the technologies related to data warehouses, business intelligence, knowledge management, and web portals.

Figure 1-15 shows an example of a dialog box displaying external meta-data related to hierarchies in MicroStrategy Desktop of the MicroStrategy 8 tool suite. In particular,

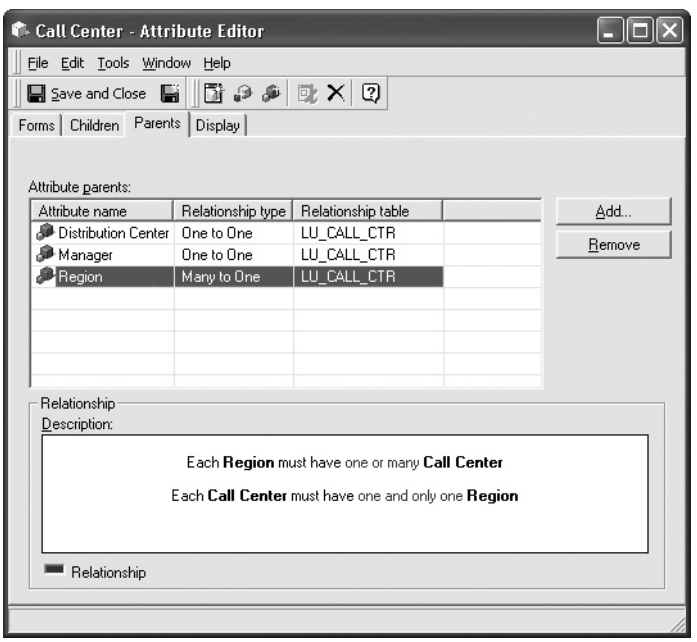


Figure 1-15 Accessing hierarchy meta-data in MicroStrategy

this dialog box displays the Calling Center attribute parent attributes. Specifically, it states that a calling center refers to a distribution center, belongs to a region, and is managed by a manager.

NOTE *See Barquin and Edelstein, 1996; Jarke et al., 2000; Jennings, 2004; and Tozer, 1999, for a comprehensive discussion on meta-data representation and management.*

1.7 Accessing Data Warehouses

Analysis is the last level common to all data warehouse architecture types. After cleansing, integrating, and transforming data, you should determine how to get the best out of it in terms of information. The following sections show the best approaches for end users to query data warehouses: *reports*, *OLAP,* and *dashboards*. End users often use the information stored to a data warehouse as a starting point for additional business intelligence applications, such as what-if analyses and data mining. See Chapter 15 for more details on these advanced applications.

1.7.1 Reports

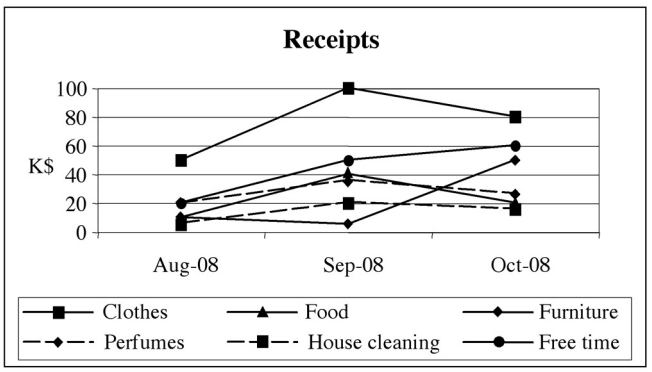
This approach is oriented to those users who need to have regular access to the information in an almost static way. For example, suppose a local health authority must send to its state offices monthly reports summing up information on patient admission costs. The layout of those reports has been predetermined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

A *report* is defined by a query and a layout. A query generally implies a restriction and an aggregation of multidimensional data. For example, you can look for the monthly receipts during the last quarter for every product category. A layout can look like a table or a chart (diagrams, histograms, pies, and so on). Figure 1-16 shows a few examples of layouts for the receipts query.

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

Keep in mind that reports existed long before data warehouse systems came to be. Reports have always been the main tool used by managers for evaluating and planning tasks since the invention of databases. However, adding data warehouses to the mix is beneficial to reports for two main reasons: First, they take advantage of reliable and correct

Receipts (K$)	Oct. 2008	Sep. 2008	Aug. 2008
Clothes	80	100	50
Food	20	40	10
Furniture	50	5	10
Perfumes	25	35	20
House cleaning	15	20	5
Free time	60	50	20



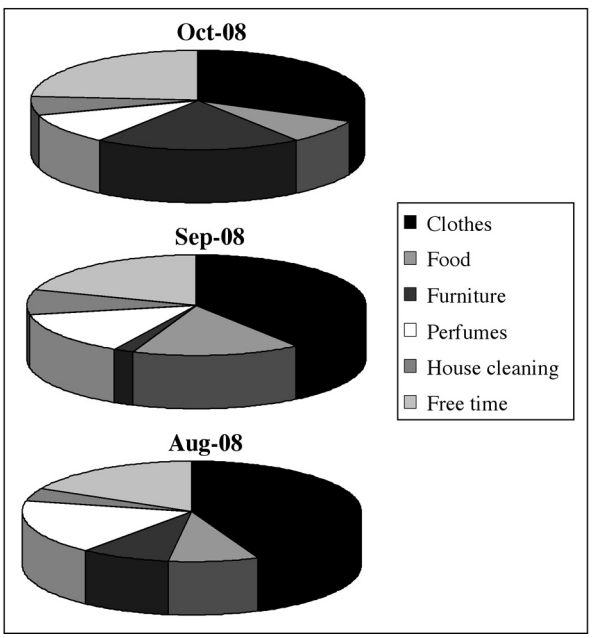


Figure 1-16 Report layouts: table (top), line graph (middle), 3-D pie graphs (bottom)

results because the data summed up in reports is consistent and integrated. In addition, data warehouses expedite the reporting process because the architectural separation between transaction processing and analyses significantly improves performance.

1.7.2 OLAP

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

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

Every step of an analysis session is characterized by an *OLAP operator* that turns the latest query into a new one. The most common operators are roll-up, drill-down, slice-and-dice, pivot, drill-across, and drill-through. The figures included here show different operators, and were generated using the MicroStrategy Desktop front-end application in the MicroStrategy 8 tool suite. They are based on the V-Mall example, in which a large virtual mall sells items from its catalog via phone and the Internet. Figure 1-17 shows the attribute hierarchies relevant to the sales fact in V-Mall.

The *roll-up* operator causes an increase in data aggregation and removes a detail level from a hierarchy. For example, Figure 1-18 shows a query posed by a user that displays

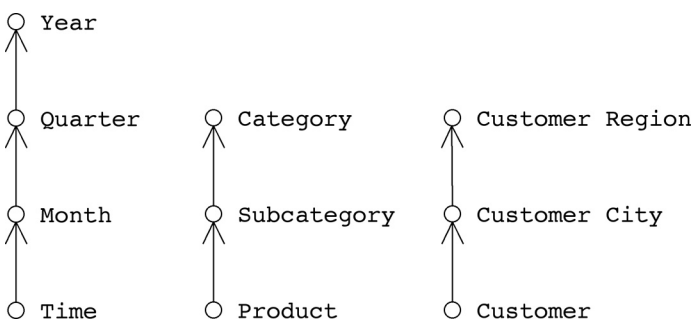


Figure 1-17 Attribute hierarchies in V-Mall; arrows show functional dependencies

Month	Metrics / Customer Region	Revenue Northeast	Mid-Atlantic	Southeast	Central	South	Northwest	Southwest
Jan 2005		$160,155	$518,405	$81,381	$322,294	$98,001	$103,368	$298,730
Feb 2005		$170,777	$491,628	$80,399	$314,466	$91,222	$114,341	$373,645
Mar 2005		$200,434	$611,424	$129,102	$382,946	$123,038	$147,472	$351,602
Apr 2005		$194,811	$502,241	$93,654	$357,188	$90,663	$124,824	$304,416
May 2005		$169,998	$462,364	$117,780	$300,389	$79,999	$117,506	$311,123
Jun 2005		$202,477	$559,466	$109,979	$309,683	$115,318	$117,008	$373,526
Jul 2005		$194,490	$577,515	$105,099	$332,300	$92,730	$103,494	$369,380
Aug 2005		$203,085	$599,761	$118,805	$410,885	$119,178	$131,148	$384,555
Sep 2005		$241,992	$625,517	$122,261	$415,763	$75,655	$124,974	$364,651
Oct 2005		$217,477	$641,340	$137,925	$382,321	$89,679	$124,276	$337,489
Nov 2005		$238,004	$708,036	$156,525	$457,105	$116,478	$156,466	$386,399
Dec 2005		$273,721	$774,372	$154,139	$479,729	$119,113	$143,753	$414,983
Jan 2006		$215,786	$662,632	$125,238	$392,922	$91,791	$122,235	$343,027
Feb 2006		$253,128	$711,937	$123,725	$415,742	$97,309	$137,589	$391,277
Mar 2006		$253,564	$704,652	$135,180	$430,143	$112,459	$144,659	$406,956
Apr 2006		$255,352	$710,402	$126,717	$426,423	$113,233	$140,976	$395,924
May 2006		$231,766	$676,205	$130,981	$440,813	$107,277	$136,043	$377,349
Jun 2006		$290,534	$769,788	$123,743	$507,166	$125,631	$131,549	$439,321
Jul 2006		$247,683	$811,060	$145,955	$448,939	$113,683	$128,113	$415,251
Aug 2006		$252,313	$719,509	$125,944	$427,188	$108,987	$153,966	$421,310
Sep 2006		$288,772	$801,819	$148,023	$539,406	$112,784	$149,236	$419,878
Oct 2006		$307,610	$710,458	$163,254	$450,006	$105,218	$144,906	$440,856
Nov 2006		$284,671	$800,941	$157,117	$505,952	$118,552	$163,560	$470,591
Dec 2006		$310,775	$891,543	$170,207	$575,086	$120,228	$155,409	$534,680

Quarter	Metrics / Customer Region	Revenue Northeast	Mid-Atlantic	Southeast	Central	South	Northwest	Southwest
2005 Q1		$531,366	$1,621,457	$290,882	$1,019,706	$312,261	$365,181	$1,023,977
2005 Q2		$567,286	$1,524,070	$321,413	$967,260	$285,981	$359,339	$989,065
2005 Q3		$639,567	$1,802,793	$346,165	$1,158,948	$287,563	$359,616	$1,118,587
2005 Q4		$729,202	$2,123,749	$448,589	$1,319,154	$325,269	$424,495	$1,138,871
2006 Q1		$722,478	$2,079,221	$384,143	$1,238,807	$301,559	$404,483	$1,141,260
2006 Q2		$777,651	$2,156,394	$381,441	$1,374,402	$346,141	$408,567	$1,212,593
2006 Q3		$788,768	$2,332,388	$419,923	$1,415,533	$335,455	$431,315	$1,256,439
2006 Q4		$903,056	$2,402,942	$490,578	$1,531,044	$343,998	$463,874	$1,446,127

Figure 1-18 Time hierarchy roll-up

monthly revenues in 2005 and 2006 for every customer region. If you "roll it up," you remove the month detail to display quarterly total revenues per region. Rolling-up can also reduce the number of dimensions in your results if you remove all the hierarchy details. If you apply this principle to Figure 1-19, you can remove information on customers and display yearly total revenues per product category as you turn the three-dimensional table

Category	Year	Metrics / Customer Region	Revenue Northeast	Mid-Atlantic	Southeast	Central	South	Northwest	Southwest
Books	2005		$416,183	$316,104	$36,517	$207,850	$137,502	$19,062	$187,368
	2006		$534,932	$401,908	$42,027	$239,806	$138,683	$22,655	$183,275
Electronics	2005		$1,860,172	$6,517,723	$1,226,825	$3,719,752	$915,633	$1,434,575	$3,625,191
	2006		$2,403,311	$8,253,620	$1,451,397	$4,631,259	$999,611	$1,615,848	$4,298,985
Movies	2005		$112,560	$138,611	$118,179	$153,556	$119,566	$27,060	$362,858
	2006		$148,785	$188,567	$147,445	$203,547	$145,434	$35,878	$463,470
Music	2005		$78,507	$99,631	$25,528	$383,911	$38,373	$27,933	$95,083
	2006		$104,925	$126,851	$35,215	$485,174	$43,424	$33,860	$110,689

Category	Year	Metrics	Revenue
Books	2005		$1,320,585
	2006		$1,563,287
Electronics	2005		$19,299,870
	2006		$23,654,030
Movies	2005		$1,032,391
	2006		$1,333,126
Music	2005		$748,966
	2006		$940,136

Figure 1-19 Roll-up removing customer hierarchy

into a two-dimensional one. Figure 1-20 uses the cube metaphor to sketch a roll-up operation with and without a decrease in dimensions.

The *drill-down* operator is the complement to the roll-up operator. Figure 1-20 shows that it reduces data aggregation and adds a new detail level to a hierarchy. Figure 1-21 shows an example based on a bidimensional table. This table shows that the aggregation based on customer regions shifts to a new fine-grained aggregation based on customer cities.

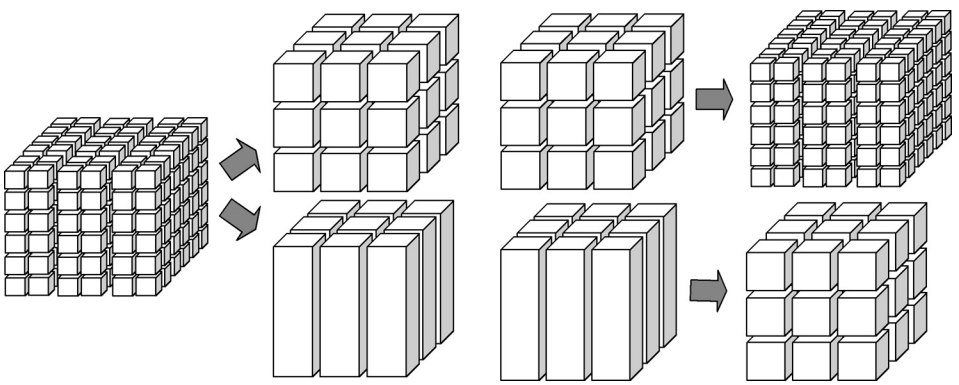


Figure 1-20 Rolling-up (left) and drilling-down (right) a cube

Quarter	Metrics / Customer Region	Revenue Northeast	Mid-Atlantic	Southeast	Central	South	Northwest	Southwest
2005 Q1		$531,366	$1,621,457	$290,882	$1,019,706	$312,261	$365,181	$1,023,977
2005 Q2		$567,286	$1,524,070	$321,413	$967,260	$285,981	$359,339	$989,065
2005 Q3		$639,567	$1,802,793	$346,165	$1,158,948	$287,563	$359,616	$1,118,587
2005 Q4		$729,202	$2,123,749	$448,589	$1,319,154	$325,269	$424,495	$1,138,871
2006 Q1		$722,478	$2,079,221	$384,143	$1,238,807	$301,559	$404,483	$1,141,260
2006 Q2		$777,651	$2,156,394	$381,441	$1,374,402	$346,141	$408,567	$1,212,593
2006 Q3		$788,768	$2,332,388	$419,923	$1,415,533	$335,455	$431,315	$1,256,439
2006 Q4		$903,056	$2,402,942	$490,578	$1,531,044	$343,998	$463,874	$1,446,127

Quarter	Metrics / Customer City	Revenue Addison	Akron	Albany	Albert City	Alexandria	Allentown	Anderson	Annapolis	Arden	Arlington Heights	Arlington	Artesia	As
2005 Q1		$7,713	$30,140	$4,626	$6,686	$29,042	$4,579	$1,948	$40,066	$23,341	$10,481		$10,922	$1
2005 Q2		$15,903	$48,029	$8,959	$2,088	$17,590	$7,268	$2,416	$42,764	$21,026	$7,514	$1,695	$2,984	$
2005 Q3		$10,091	$30,510	$5,763	$11,380	$26,389	$10,195	$568	$51,650	$25,132	$10,784	$3,796	$8,701	$
2005 Q4		$12,425	$48,588	$10,939	$10,463	$28,016	$10,426	$3,412	$67,515	$28,398	$14,692		$9,975	$1
2006 Q1		$7,256	$26,183	$7,998	$5,603	$35,959	$10,273	$2,732	$50,121	$26,351	$7,276	$1,593	$6,208	$1
2006 Q2		$10,411	$49,540	$6,065	$5,670	$25,166	$6,992	$1,377	$68,198	$27,556	$16,755	$3,420	$6,656	$1
2006 Q3		$10,325	$38,414	$9,108	$7,760	$33,170	$16,978	$821	$69,858	$33,579	$10,235	$4,319	$7,636	$1
2006 Q4		$16,613	$49,133	$13,137	$10,637	$30,344	$13,875	$747	$48,305	$32,842	$13,311	$6,176	$9,191	$1

FIGURE 1-21 Drilling-down customer hierarchy

In Figure 1-22, the drill-down operator causes an increase in the number of table dimensions after adding customer region details.

Slice-and-dice is one of the most abused terms in data warehouse literature because it can have many different meanings. A few authors use it generally to define the whole OLAP navigation process. Other authors use it to define selection and projection operations based on data. In compliance with section 1.5.1, we define *slicing* as an operation that reduces the

Category	Metrics / Year	Revenue 2005	2006
Books		$1,320,585	$1,563,287
Electronics		$19,299,870	$23,654,030
Movies		$1,032,391	$1,333,126
Music		$748,966	$940,136

Category	Metrics / Customer Region / Year	Revenue Northeast 2005	2006	Mid-Atlantic 2005	2006	Southeast 2005	2006	Central 2005	2006	South 2005	2006	Nort 2
Books		$416,183	$534,932	$316,104	$401,908	$36,517	$42,027	$207,850	$239,806	$137,502	$138,683	$
Electronics		$1,860,172	$2,403,311	$6,517,723	$8,253,620	$1,226,825	$1,451,397	$3,719,752	$4,631,259	$915,633	$999,611	$1,4
Movies		$112,560	$148,785	$138,611	$188,567	$118,179	$147,445	$153,556	$203,547	$119,566	$145,434	$
Music		$78,507	$104,925	$99,631	$126,851	$25,528	$35,215	$383,911	$485,174	$38,373	$43,424	$

FIGURE 1-22 Drilling-down and adding a dimension

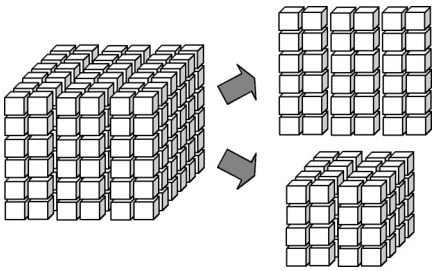


FIGURE 1-23 Slicing (above) and dicing (below) a cube

number of cube dimensions after setting one of the dimensions to a specific value. *Dicing* is an operation that reduces the set of data being analyzed by a selection criterion (Figure 1-23). Figures 1-24 and 1-25 show a few examples of slicing and dicing.

The *pivot* operator implies a change in layouts. It aims at analyzing an individual group of information from a different viewpoint. According to the multidimensional metaphor, if you pivot data, you rotate your cube so that you can rearrange cells on the

Category	Year	Metrics / Customer Region	Revenue Northeast	Mid-Atlantic	Southeast	Central	South	Northwest	Southwest
Books	2005		$416,183	$316,104	$36,517	$207,850	$137,502	$19,062	$187,368
	2006		$534,932	$401,908	$42,027	$239,806	$138,683	$22,655	$183,275
Electronics	2005		$1,860,172	$6,517,723	$1,226,825	$3,719,752	$915,633	$1,434,575	$3,625,191
	2006		$2,403,311	$8,253,620	$1,451,397	$4,631,259	$999,611	$1,615,848	$4,298,985
Movies	2005		$112,560	$138,611	$118,179	$153,556	$119,566	$27,060	$362,858
	2006		$148,785	$188,567	$147,445	$203,547	$145,434	$35,878	$463,470
Music	2005		$78,507	$99,631	$25,528	$383,911	$38,373	$27,933	$95,083
	2006		$104,925	$126,851	$35,215	$485,174	$43,424	$33,860	$110,689

Report Filter (Local Filter):
Year = 2006

Category	Metrics / Customer Region	Revenue Northeast	Mid-Atlantic	Southeast	Central	South	Northwest	Sout
Books		$534,932	$401,908	$42,027	$239,806	$138,683	$22,655	$1
Electronics		$2,403,311	$8,253,620	$1,451,397	$4,631,259	$999,611	$1,615,848	$4,2
Movies		$148,785	$188,567	$147,445	$203,547	$145,434	$35,878	$4
Music		$104,925	$126,851	$35,215	$485,174	$43,424	$33,860	$1

FIGURE 1-24 Slicing based on the `Year`='2006' predicate

Metrics / Customer City / Subcategory	Revenue: Addison	Akron	Albany	Albert City	Alexandria	Allentown	Anderson	Annapolis	Arden	Arlin Heig
Art & Architecture	$253	$365	$1,506	$279	$268	$1,960		$407	$282	
Business	$198	$357	$719	$75	$134	$1,225	$8	$304	$184	
Literature	$92	$116	$277	$54	$66	$503		$137	$128	
Books - Miscellaneous	$216	$95	$830	$120	$73	$605	$4	$233	$71	
Science & Technology	$363	$943	$3,271	$578	$491	$2,547		$834	$366	
Sports & Health	$220	$153	$416	$165	$128	$1,476		$409	$204	
Audio Equipment	$4,782	$32,192	$2,982	$4,318	$36,458	$7,303	$4,430	$77,591	$17,397	$4
Cameras	$7,671	$39,381	$5,810	$3,898	$12,045	$8,124	$405	$33,744	$11,840	$6
Computers	$1,531	$14,810	$1,935	$3,022	$7,967	$3,354		$5,097	$7,087	$2
Electronics - Miscellaneous	$4,667	$25,861	$5,289	$5,979	$25,484	$4,821	$130	$9,533	$11,923	$6
TV's	$9,027	$34,635	$5,013	$3,500	$33,432	$4,930		$46,933	$32,527	$1
Video Equipment	$7,422	$10,078	$5,540	$2,479	$5,500	$7,470	$540	$55,552	$35,723	$8
Action	$108	$302	$204	$134	$256	$259	$25	$859	$117	
Comedy	$308	$520	$355	$195	$204	$261		$361	$216	
Drama	$424	$548	$225	$243	$313	$399		$511	$415	
Horror	$339	$196	$141	$195	$188	$372		$261	$184	
Kids / Family	$361	$277	$236	$213	$235	$460	$11	$744	$323	
Special Interests	$700	$670	$353	$289	$179	$695		$816	$288	
Alternative	$1,077	$236	$114	$484	$234	$139		$365	$173	$1
Country	$1,169	$336	$310	$576	$206	$123	$43	$283	$288	
Music - Miscellaneous	$1,193	$364	$167	$494	$254	$281	$18	$410	$132	$1
Pop	$541	$286	$231	$392	$139	$178	$26	$350	$173	
Rock	$1,184	$324	$202	$490	$161	$341	$14	$285	$127	
Soul / R&B	$760	$225	$182	$1,496	$223	$293	$23	$463	$159	$1

Report Filter (Local Filter):
(Year = 2006) And (Category = Electronics) And (Revenue > 80) And ({Customer Region} = Northwest)

Metrics / Customer City / Subcategory	Revenue: Bellevue	Bothell	Caldwell	Cheyenne	Coulee City	Eu
Audio Equipment	$9,945	$23,211	$2,531	$2,347	$30,857	$
Cameras	$31,245	$5,613	$5,592	$1,240	$11,250	$
Computers	$12,751	$4,771	$1,443	$713	$8,181	
Electronics - Miscellaneous	$13,940	$29,072	$3,722	$2,444	$16,254	$
TV's	$40,652	$9,788	$3,990	$1,871	$10,846	$
Video Equipment	$36,423	$26,363	$3,543	$3,947	$5,480	$

Figure 1-25 Selection based on a complex predicate

basis of a new perspective. In practice, you can highlight a different combination of dimensions (Figure 1-26). Figures 1-27 and 1-28 show a few examples of pivoted two-dimensional and three-dimensional tables.

The term *drill-across* stands for the opportunity to create a link between two or more interrelated cubes in order to compare their data. For example, this applies if you calculate

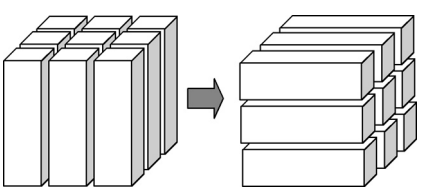


Figure 1-26
Pivoting a cube

Category	Year	Metrics	Revenue
Books	2005		$1,320,585
	2006		$1,563,287
Electronics	2005		$19,299,870
	2006		$23,654,030
Movies	2005		$1,032,391
	2006		$1,333,126
Music	2005		$748,966
	2006		$940,136

Category	Metrics / Year	Revenue 2005	Revenue 2006
Books		$1,320,585	$1,563,287
Electronics		$19,299,870	$23,654,030
Movies		$1,032,391	$1,333,126
Music		$748,966	$940,136

FIGURE 1-27 Pivoting a two-dimensional table

an expression involving measures from two cubes (Figure 1-29). Figure 1-30 shows an example in which a sales cube is drilled-across a promotions cube in order to compare revenues and discounts per quarter and product category.

Most OLAP tools can perform *drill-through* operations, though with varying effectiveness. This operation switches from multidimensional aggregate data in data marts to operational data in sources or in the reconciled layer.

In many applications, an intermediate approach between static reporting and OLAP is broadly used. This intermediate approach is called *semi-static reporting*. Even if a semi-static report focuses on a group of information previously set, it gives users some margin of freedom. Thanks to this margin, users can follow a limited set of navigation paths. For example, this applies when you can roll up just to a few hierarchy attributes. This solution is common, because it provides some unquestionable advantages. First, users need less skill to use data models and analysis tools than they need for OLAP. Second, this avoids the risk that occurs in OLAP of achieving inconsistent analysis results or incorrect ones because of any misuse of aggregation operators. Third, if you pose constraints on the analyses allowed, you will prevent users from unwillingly slowing down your system whenever they formulate demanding queries.

Category	Year	Metrics / Customer Region	Revenue Northeast	Mid-Atlantic	Southeast	Central	South	Northwest	Southwest
Books	2005		$416,183	$316,104	$36,517	$207,850	$137,502	$19,062	$187,368
	2006		$534,932	$401,908	$42,027	$239,806	$138,683	$22,655	$183,275
Electronics	2005		$1,860,172	$6,517,723	$1,226,825	$3,719,752	$915,633	$1,434,575	$3,625,191
	2006		$2,403,311	$8,253,620	$1,451,397	$4,631,259	$999,611	$1,615,848	$4,298,985
Movies	2005		$112,560	$138,611	$118,179	$153,556	$119,566	$27,060	$362,858
	2006		$148,785	$188,567	$147,445	$203,547	$145,434	$35,878	$463,470
Music	2005		$78,507	$99,631	$25,528	$383,911	$38,373	$27,933	$95,083
	2006		$104,925	$126,851	$35,215	$485,174	$43,424	$33,860	$110,689

Metrics / Customer Region / Year / Category	Revenue Northeast 2005	Northeast 2006	Mid-Atlantic 2005	Mid-Atlantic 2006	Southeast 2005	Southeast 2006	Central 2005	Central 2006	South 2005	South 2006	Nort 2
Books	$416,183	$534,932	$316,104	$401,908	$36,517	$42,027	$207,850	$239,806	$137,502	$138,683	$
Electronics	$1,860,172	$2,403,311	$6,517,723	$8,253,620	$1,226,825	$1,451,397	$3,719,752	$4,631,259	$915,633	$999,611	$1,4
Movies	$112,560	$148,785	$138,611	$188,567	$118,179	$147,445	$153,556	$203,547	$119,566	$145,434	$
Music	$78,507	$104,925	$99,631	$126,851	$25,528	$35,215	$383,911	$485,174	$38,373	$43,424	$

FIGURE 1-28 Pivoting a three-dimensional table

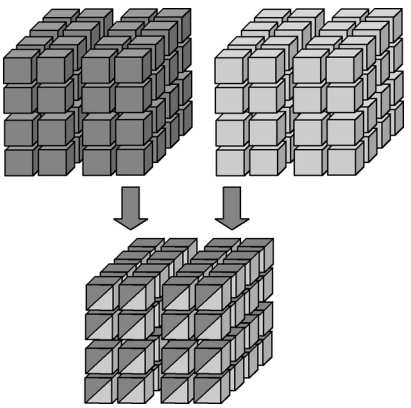


Figure 1-29 Drilling across two cubes

1.7.3 Dashboards

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

Today, most software vendors offer dashboards for report creation and display. Figure 1-31 shows a dashboard created with MicroStrategy Dynamic Enterprise. The literature related to dashboard graphic design has also proven to be very rich, in particular in the scope of enterprises (Few, 2006).

Category	Metrics / Quarter	Revenue 2005 Q1	2005 Q2	2005 Q3	2005 Q4	2006 Q1	2006 Q2	2006 Q3	2006 Q4
Books		$319,767	$313,339	$336,862	$350,617	$348,483	$387,849	$407,392	$419,563
Electronics		$4,448,112	$4,299,411	$4,918,673	$5,633,676	$5,411,499	$5,714,783	$5,999,174	$6,528,576
Movies		$228,108	$232,201	$264,471	$307,611	$299,531	$326,270	$334,143	$373,182
Music		$168,843	$169,462	$193,234	$217,427	$212,438	$228,289	$239,112	$260,298

Category	Quarter / Metrics	2005 Q1 Discount	2005 Q1 Revenue	2005 Q2 Discount	2005 Q2 Revenue	2005 Q3 Discount	2005 Q3 Revenue	2005 Q4 Discount	2005 Q4 Revenue	2006 Q1 Discount	2006 Q1 Re
Books		$ 0	$319,767	$ 10,845	$313,339	$ 9,497	$336,862	$ 18,279	$350,617	$ 0	
Electronics		$ 0	$4,448,112	$ 150,366	$4,299,410	$ 143,395	$4,918,673	$ 302,884	$5,633,675	$ 0	$5
Movies		$ 0	$228,108	$ 8,025	$232,201	$ 7,948	$264,471	$ 16,649	$307,611	$ 0	
Music		$ 0	$168,843	$ 6,143	$169,462	$ 5,563	$193,234	$ 11,047	$217,427	$ 0	

Figure 1-30 Drilling across the sales cube (`Revenue` measure) and the promotions cube (`Discount` measure)

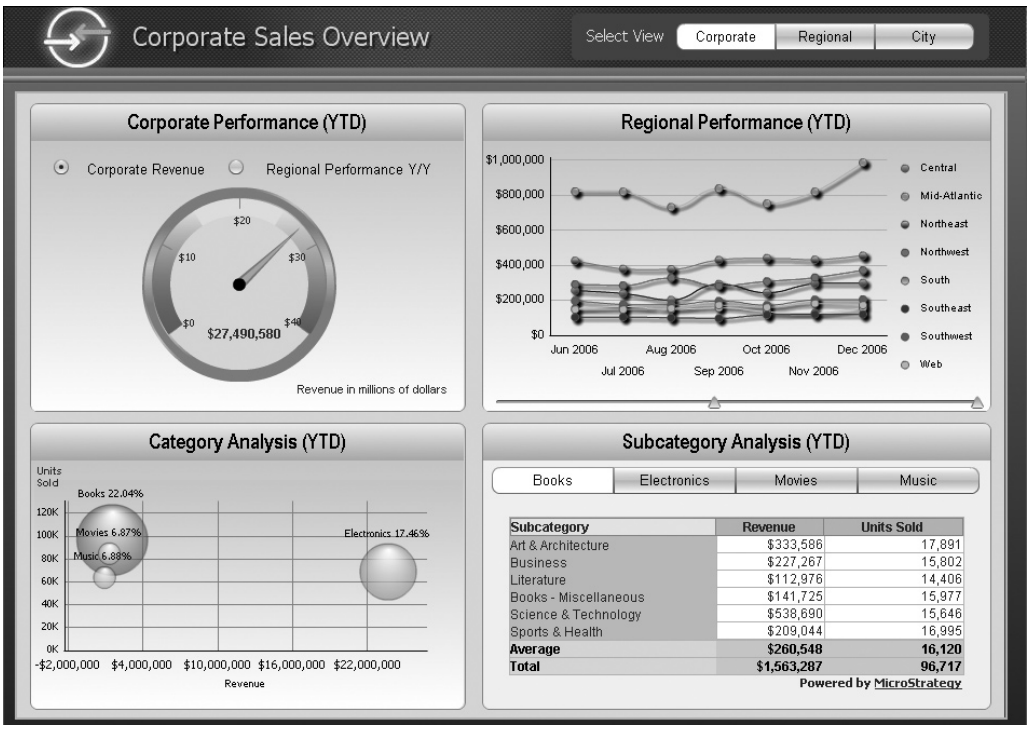


FIGURE 1-31 An example of dashboards

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

1.8 ROLAP, MOLAP, and HOLAP

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

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

The idea of adopting the relational technology to store data to a data warehouse has a solid foundation if you consider the huge amount of literature written about the relational model, the broadly available corporate experience with relational database usage and management, and the top performance and flexibility standards of relational DBMSs (RDBMSs). The expressive power of the relational model, however, does not include the concepts of dimension, measure, and hierarchy, so you must create specific types of schemata so that you can represent the multidimensional model in terms of basic relational elements such as attributes, relations, and integrity constraints. This task is mainly performed by the well-known *star schema*. See Chapter 8 for more details on star schemata and star schema variants.

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

From an architectural viewpoint, adopting ROLAP requires specialized *middleware,* also called a *multidimensional engine*, between relational back-end servers and front-end components, as shown in Figure 1-32. The middleware receives OLAP queries formulated by users in a front-end tool and turns them into SQL instructions for a relational back-end application with the support of meta-data. The so-called *aggregate navigator* is a particularly important component in this phase. In case of aggregate views, this component selects a view from among all the alternatives to solve a specific query at the minimum access cost.

In commercial products, different front-end modules, such as OLAP, reports, and dashboards, are generally strictly connected to a multidimensional engine. Multidimensional engines are the main components and can be connected to any relational server. Open source solutions have been recently released. Their multidimensional engines (Mondrian, 2009) are disconnected from front-end modules (JPivot, 2009). For this reason, they can be more flexible

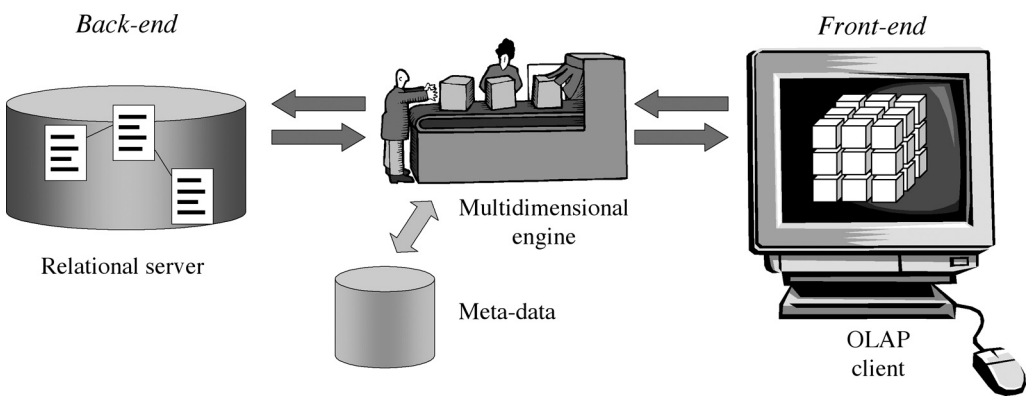


Figure 1-32 ROLAP architecture

than commercial solutions when you have to create the architecture (Thomsen and Pedersen, 2005). A few commercial RDBMSs natively support features typical for multidimensional engines to maximize query optimization and increase meta-data reusability. For example, since its 8*i* version was made available, Oracle's RDBMS gives users the opportunity to define hierarchies and materialized views. Moreover, it offers a navigator that can use meta-data and rewrite queries without any need for a multidimensional engine to be involved.

Different from a ROLAP system, a MOLAP system is based on an ad hoc logical model that can be used to represent multidimensional data and operations directly. The underlying multidimensional database physically stores data as arrays and the access to it is positional (Gaede and Günther, 1998). *Grid-files* (Nievergelt et al., 1984; Whang and Krishnamurthy, 1991), *R*-trees* (Beckmann et al., 1990) and *UB-trees* (Markl et al., 2001) are among the techniques used for this purpose.

The greatest advantage of MOLAP systems in comparison with ROLAP is that multidimensional operations can be performed in an easy, natural way with MOLAP without any need for complex join operations. For this reason, MOLAP system performance is excellent. However, MOLAP system implementations have very little in common, because no multidimensional logical model standard has yet been set. Generally, they simply share the usage of optimization techniques specifically designed for sparsity management. The lack of a common standard is a problem being progressively solved. This means that MOLAP tools are becoming more and more successful after their limited implementation for many years. This success is also proven by the investments in this technology by major vendors, such as Microsoft (Analysis Services) and Oracle (Hyperion).

The intermediate architecture type, *HOLAP*, aims at mixing the advantages of both basic solutions. It takes advantage of the standardization level and the ability to manage large amounts of data from ROLAP implementations, and the query speed typical of MOLAP systems. HOLAP implies that the largest amount of data should be stored in an RDBMS to avoid the problems caused by sparsity, and that a multidimensional system stores only the information users most frequently need to access. If that information is not enough to solve queries, the system will transparently access the part of the data managed by the relational system. Over the last few years, important market actors such as MicroStrategy have adopted HOLAP solutions to improve their platform performance, joining other vendors already using this solution, such as Business Objects.

1.9 Additional Issues

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

1.9.1 Quality

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

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

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

Technically, checking for data quality requires appropriate sets of metrics (Abelló et al., 2006). In the following sections, we provide an example of the metrics for a few of the quality properties mentioned:

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

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

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

NOTE *Jarke et al., 2000 have closely studied data quality. Their studies provide useful discussions on the impact of data quality problems from the methodological point of view. Kelly, 1997 describes quality goals strictly connected to the viewpoint of business organizations. Serrano et al., 2004, 2007; Lechtenbörger, 2001; and Bouzeghoub and Kedad, 2000 focus on quality standards respectively for conceptual, logical, and physical data warehouse schemata.*

1.9.2 Security

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

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

NOTE *In the scientific literature there are a few works specifically dealing with security in data warehouse systems (Kirkgöze et al., 1997; Priebe and Pernul, 2000; Rosenthal and Sciore, 2000; Katic et al., 1998). In particular, Priebe and Pernul propose a comparative study on security properties of a few commercial platforms. Ferrandez-Medina et al., 2004 and Soler et al., 2008 discuss an approach that could be more interesting for designers. They use a UML extension to model specific security requirements for data warehouses in the conceptual design and requirement analysis phases, respectively.*

1.9.3 Evolution

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

- **Data level** Even if measured data is naturally logged in data warehouses thanks to temporal dimensions marking events, the multidimensional model implicitly assumes that hierarchies are completely static. It is clear that this assumption is not very realistic. For example, a company can add new product categories to its catalog and remove others, or it can change the category to which an existing product belongs in order to meet new marketing strategies.

- **Schema level** A data warehouse schema can vary to meet new business domain standards, new users' requirements, or changes in data sources. New attributes and measures can become necessary. For example, you can add a subcategory to a product hierarchy to make analyses richer in detail. You should also consider that the set of fact dimensions can vary as time goes by.

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

As far as changes in data values are concerned, different approaches have been documented in scientific literature. Some commercial systems also make it possible to track changes and query cubes on the basis of different temporal scenarios. See section 8.4 for more details on dynamic hierarchies. On the other hand, managing changes in data schemata has been explored only partially to date. No commercial tool is currently available on the market to support approaches to data schema change management.

The approaches to data warehouse schema change management can be classified in two categories: *evolution* (Quix, 1999; Vaisman et al., 2002; Blaschka, 2000) and *versioning* (Eder et al., 2002; Golfarelli et al., 2006a). Both categories make it possible to alter data schemata, but only versioning can track previous schema releases. A few approaches to versioning can create not only "true" versions generated by changes in application domains, but also alternative versions to use for what-if analyses (Bebel et al., 2004).

The main problem that has not been solved in this field is the creation of techniques for versioning and data migration between versions that can flexibly support queries related to more schema versions. Furthermore, we need systems that can semiautomatically adjust ETL procedures to changes in source schemata. In this direction, some OLAP tools already use their meta-data to support an impact analysis aimed at identifying the full consequences of any changes in source schemata.