Contents

Chapter 1

Introduction

This book is an introduction to what has come to be known as *data mining* and *knowledge discovery in databases*. The material in this book is presented from a database perspective, where emphasis is placed on basic data mining concepts and techniques for uncovering interesting data patterns hidden in *large data sets*. The implementation methods discussed are particularly oriented towards the development of *scalable* and *efficient* data mining tools.

In this chapter, you will learn how data mining is part of the natural evolution of database technology, why data mining is important, and how it is defined. You will learn about the general architecture of data mining systems, as well as gain insight into the kinds of data on which mining can be performed, the types of patterns that can be found, and how to tell which patterns represent useful knowledge. In addition to studying a classification of data mining systems, you will read about challenging research issues for building data mining tools of the future.

1.1 What motivated data mining? Why is it important?

Necessity is the mother of invention.

— *English proverb.*

The major reason that data mining has attracted a great deal of attention in information industry in recent years is due to the wide availability of huge amounts of data and the imminent need for turning such data into useful information and knowledge. The information and knowledge gained can be used for applications ranging from business management, production control, and market analysis, to engineering design and science exploration.

Data mining can be viewed as a result of the natural evolution of information technology. An evolutionary path has been witnessed in the database industry in the development of the following functionalities (Figure 1.1): *data collection and database creation, data management* (including data storage and retrieval, and database transaction processing), and *data analysis and understanding* (involving data warehousing and data mining). For instance, the early development of data collection and database creation mechanisms served as a prerequisite for later development of effective mechanisms for data storage and retrieval, and query and transaction processing. With numerous database systems offering query and transaction processing as common practice, data analysis and understanding has naturally become the next target.

Since the 1960's, database and information technology has been evolving systematically from primitive file processing systems to sophisticated and powerful databases systems. The research and development in database systems since the 1970's has led to the development of relational database systems (where data are stored in relational table structures; see Section 1.3.1), data modeling tools, and indexing and data organization techniques. In addition, users gained convenient and flexible data access through query languages, query processing, and user interfaces. Efficient methods for **on-line transaction processing** (OLTP), where a query is viewed as a read-only transaction, have contributed substantially to the evolution and wide acceptance of relational technology as a major tool for efficient storage, retrieval, and management of large amounts of data.

Database technology since the mid-1980s has been characterized by the popular adoption of relational technology and an upsurge of research and development activities on new and powerful database systems. These employ ad-

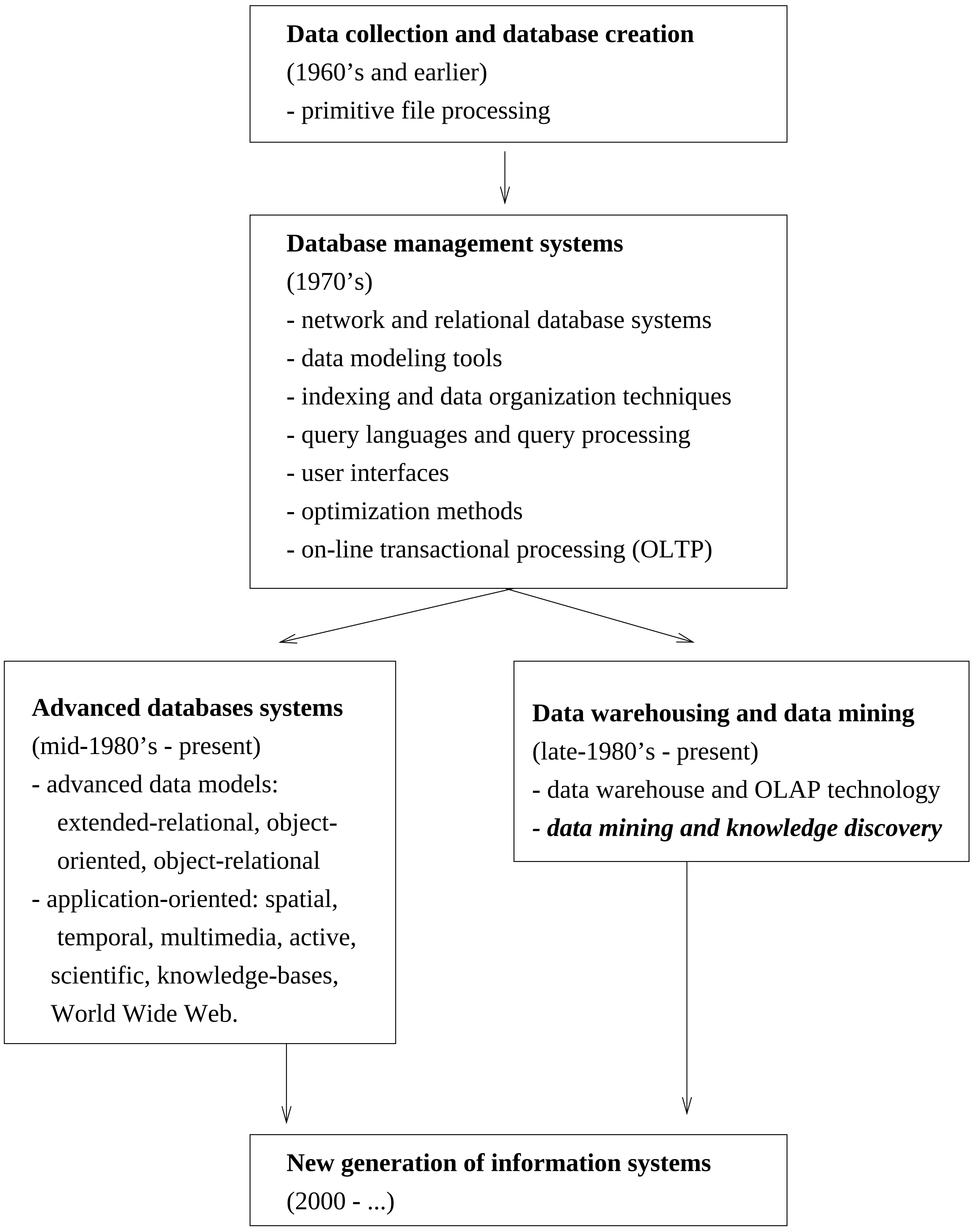

Data collection and database creation
(1960's and earlier)
- primitive file processing
Database management systems
(1970's)
- network and relational database systems
- data modeling tools
- indexing and data organization techniques
- query languages and query processing
- user interfaces
- optimization methods
- on-line transactional processing (OLTP)
Advanced databases systems
(mid-1980's - present)
- advanced data models: extended-relational, object-oriented, object-relational
- application-oriented: spatial, temporal, multimedia, active, scientific, knowledge-bases, World Wide Web.
Data warehousing and data mining
(late-1980's - present)
- data warehouse and OLAP technology
- data mining and knowledge discovery
New generation of information systems
(2000 - ...)

Figure 1.1: The evolution of database technology.

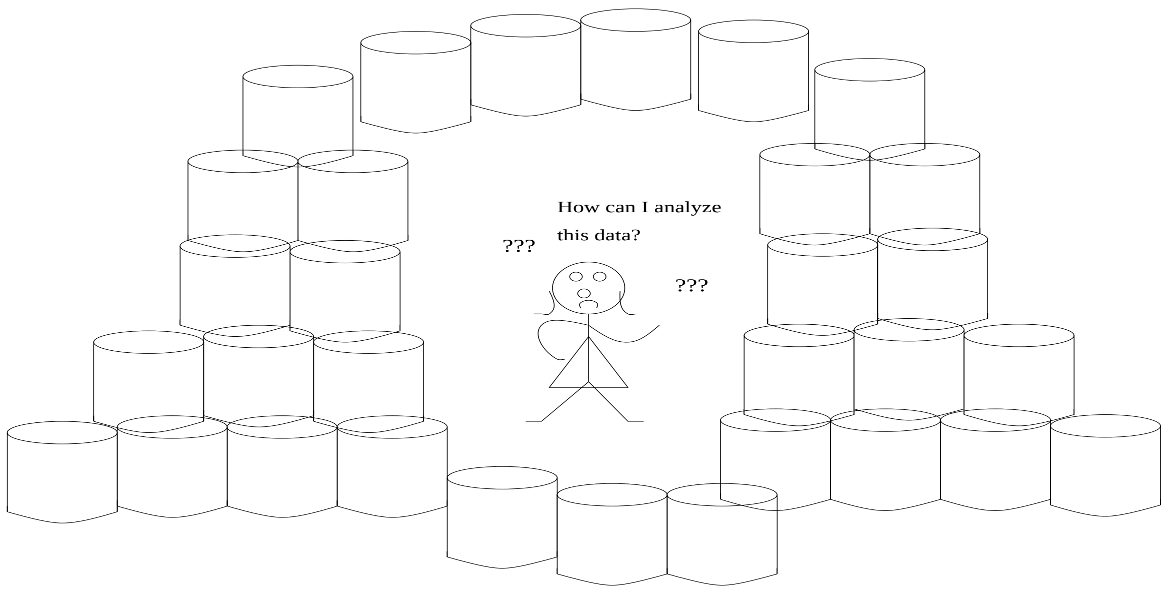


Figure 1.2: We are data rich, but information poor.

vanced data models such as extended-relational, object-oriented, object-relational, and deductive models Application-oriented database systems, including spatial, temporal, multimedia, active, and scientific databases, knowledge bases, and office information bases, have flourished. Issues related to the distribution, diversification, and sharing of data have been studied extensively. Heterogeneous database systems and Internet-based global information systems such as the World-Wide Web (WWW) also emerged and play a vital role in the information industry.

The steady and amazing progress of computer hardware technology in the past three decades has led to powerful, affordable, and large supplies of computers, data collection equipment, and storage media. This technology provides a great boost to the database and information industry, and makes a huge number of databases and information repositories available for transaction management, information retrieval, *and data analysis*.

Data can now be stored in many different types of databases. One database architecture that has recently emerged is the **data warehouse** (Section 1.3.2), a repository of multiple heterogeneous data sources, organized under a unified schema at a single site in order to facilitate management decision making. Data warehouse technology includes data cleansing, data integration, and **On-Line Analytical Processing (OLAP)**, that is, analysis techniques with functionalities such as summarization, consolidation and aggregation, as well as the ability to view information at different angles. Although OLAP tools support multidimensional analysis and decision making, additional data analysis tools are required for in-depth analysis, such as data classification, clustering, and the characterization of data changes over time.

The abundance of data, coupled with the need for powerful data analysis tools, has been described as a "*data rich but information poor*" situation. The fast-growing, tremendous amount of data, collected and stored in large and numerous databases, has far exceeded our human ability for comprehension *without powerful tools* (Figure 1.2). As a result, data collected in large databases become "data tombs" — data archives that are seldom revisited. Consequently, important decisions are often made based not on the information-rich data stored in databases but rather on a decision maker's intuition, simply because the decision maker does not have the tools to extract the valuable knowledge embedded in the vast amounts of data. In addition, consider current expert system technologies, which typically rely on users or domain experts to *manually* input knowledge into knowledge bases. Unfortunately, this procedure is prone to biases and errors, and is extremely time-consuming and costly. Data mining tools which perform data analysis may uncover important data patterns, contributing greatly to business strategies, knowledge bases, and scientific and medical research. The widening gap between data and information calls for a systematic development of *data mining tools* which will turn data tombs into "golden nuggets" of knowledge.

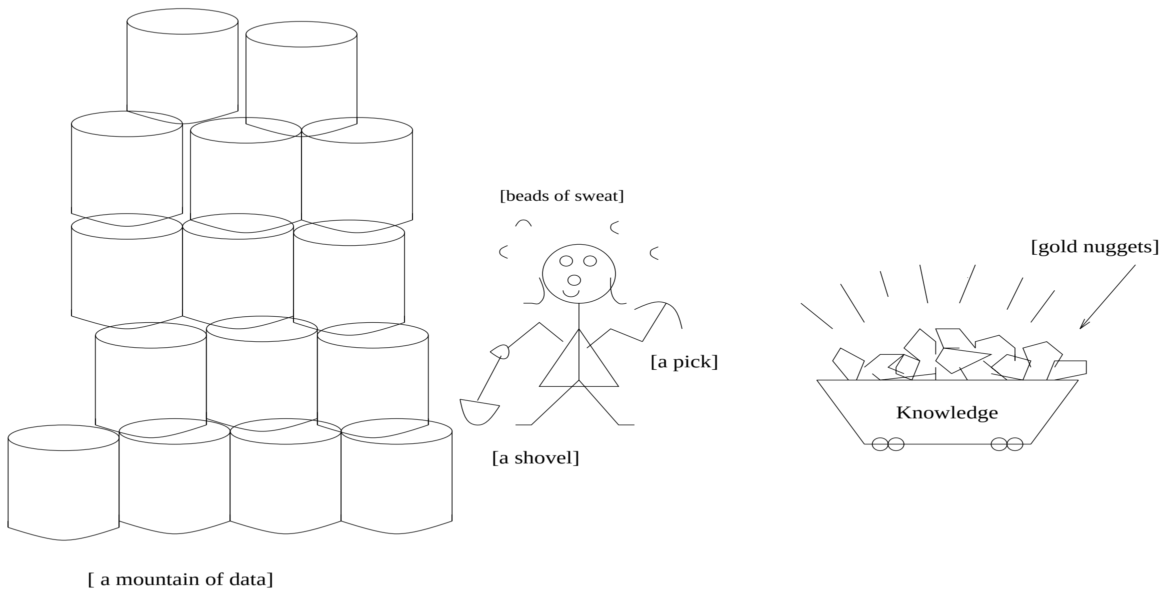


Figure 1.3: Data mining - searching for knowledge (interesting patterns) in your data.

1.2 So, what is data mining?

Simply stated, **data mining** refers to *extracting or "mining" knowledge from large amounts of data.* The term is actually a misnomer. Remember that the mining of gold from rocks or sand is referred to as *gold* mining rather than rock or sand mining. Thus, "data mining" should have been more appropriately named "knowledge mining from data", which is unfortunately somewhat long. "Knowledge mining", a shorter term, may not reflect the emphasis on mining from large amounts of data. Nevertheless, mining is a vivid term characterizing the process that finds a small set of precious nuggets from a great deal of raw material (Figure 1.3). Thus, such a misnomer which carries both "data" and "mining" became a popular choice. There are many other terms carrying a similar or slightly different meaning to data mining, such as **knowledge mining from databases, knowledge extraction, data/pattern analysis, data archaeology**, and **data dredging**.

Many people treat data mining as a synonym for another popularly used term, "**Knowledge Discovery in Databases**", or **KDD**. Alternatively, others view data mining as simply an essential step in the process of knowledge discovery in databases. Knowledge discovery as a process is depicted in Figure 1.4, and consists of an iterative sequence of the following steps:

- **data cleaning** (to remove noise or irrelevant data),
- **data integration** (where multiple data sources may be combined)[1],
- **data selection** (where data relevant to the analysis task are retrieved from the database),
- **data transformation** (where data are transformed or consolidated into forms appropriate for mining by performing summary or aggregation operations, for instance)[2],
- **data mining** (an essential process where intelligent methods are applied in order to extract data patterns),
- **pattern evaluation** (to identify the truly interesting patterns representing knowledge based on some **interestingness measures**; Section 1.5), and
- **knowledge presentation** (where visualization and knowledge representation techniques are used to present the mined knowledge to the user).

[1]A popular trend in the information industry is to perform data cleaning and data integration as a preprocessing step where the resulting data are stored in a data warehouse.

[2]Sometimes data transformation and consolidation are performed before the data selection process, particularly in the case of data warehousing.

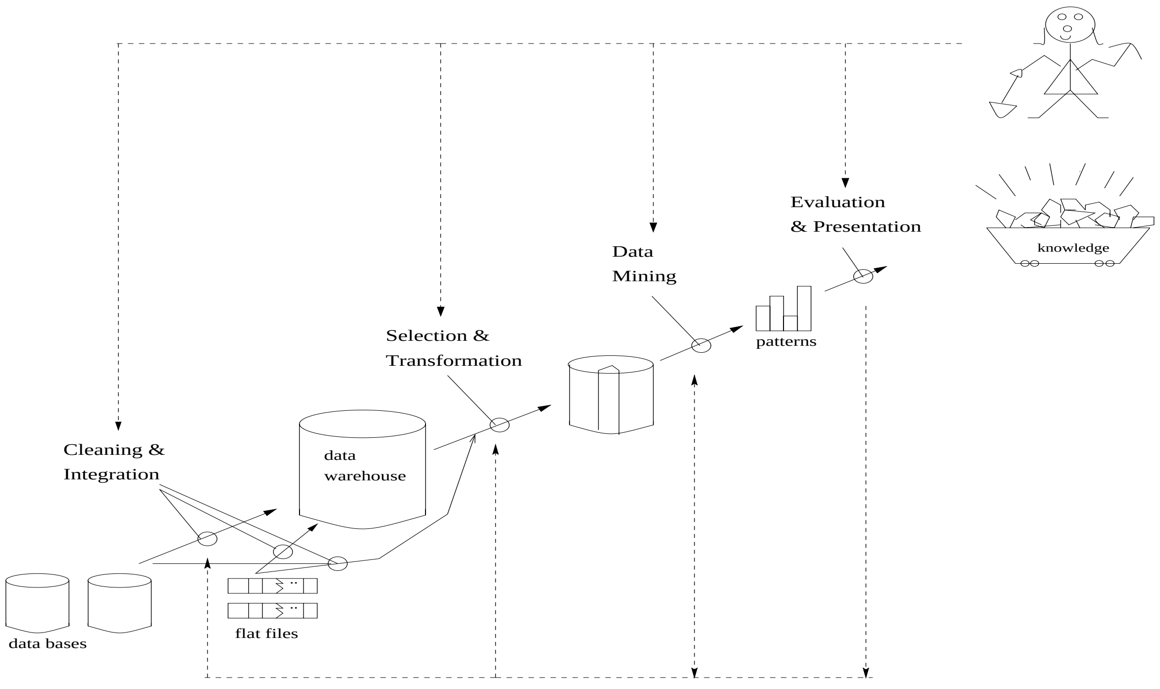


Figure 1.4: Data mining as a process of knowledge discovery.

The data mining step may interact with the user or a knowledge base. The interesting patterns are presented to the user, and may be stored as new knowledge in the knowledge base. Note that according to this view, data mining is only one step in the entire process, albeit an essential one since it uncovers hidden patterns for evaluation.

We agree that data mining is a knowledge discovery process. However, in industry, in media, and in the database research milieu, the term "data mining" is becoming more popular than the longer term of "knowledge discovery in databases". Therefore, in this book, we choose to use the term "data mining". We adopt a broad view of data mining functionality: **data mining** is the process of discovering interesting knowledge from *large* amounts of data stored either in databases, data warehouses, or other information repositories.

Based on this view, the architecture of a typical data mining system may have the following major components (Figure 1.5):

1. **Database, data warehouse, or other information repository.** This is one or a set of databases, data warehouses, spread sheets, or other kinds of information repositories. Data cleaning and data integration techniques may be performed on the data.

2. **Database or data warehouse server.** The database or data warehouse server is responsible for fetching the relevant data, based on the user's data mining request.

3. **Knowledge base.** This is the domain knowledge that is used to guide the search, or evaluate the interestingness of resulting patterns. Such knowledge can include **concept hierarchies**, used to organize attributes or attribute values into different levels of abstraction. Knowledge such as user beliefs, which can be used to assess a pattern's interestingness based on its unexpectedness, may also be included. Other examples of domain knowledge are additional interestingness constraints or thresholds, and metadata (e.g., describing data from multiple heterogeneous sources).

4. **Data mining engine.** This is essential to the data mining system and ideally consists of a set of functional modules for tasks such as characterization, association analysis, classification, evolution and deviation analysis.

5. **Pattern evaluation module.** This component typically employs interestingness measures (Section 1.5) and interacts with the data mining modules so as to *focus* the search towards interesting patterns. It may access interestingness thresholds stored in the knowledge base. Alternatively, the pattern evaluation module may be

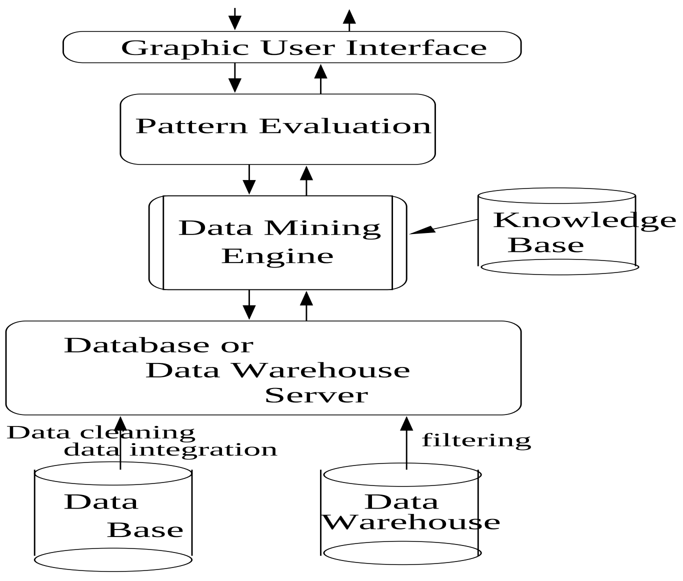


Figure 1.5: Architecture of a typical data mining system.

integrated with the mining module, depending on the implementation of the data mining method used. For efficient data mining, it is highly recommended to push the evaluation of pattern interestingness as deep as possible into the mining process so as to confine the search to only the interesting patterns.

6. **Graphical user interface.** This module communicates between users and the data mining system, allowing the user to interact with the system by specifying a data mining query or task, providing information to help focus the search, and performing exploratory data mining based on the intermediate data mining results. In addition, this component allows the user to browse database and data warehouse schemas or data structures, evaluate mined patterns, and visualize the patterns in different forms.

From a data warehouse perspective, data mining can be viewed as an advanced stage of on-line analytical processing (OLAP). However, data mining goes far beyond the narrow scope of summarization-style analytical processing of data warehouse systems by incorporating more advanced techniques for data understanding.

While there may be many "data mining systems" on the market, not all of them can perform true data mining. A data analysis system that does not handle large amounts of data can at most be categorized as a machine learning system, a statistical data analysis tool, or an experimental system prototype. A system that can only perform data or information retrieval, including finding aggregate values, or that performs deductive query answering in large databases should be more appropriately categorized as either a database system, an information retrieval system, or a deductive database system.

Data mining involves an integration of techniques from multiple disciplines such as database technology, statistics, machine learning, high performance computing, pattern recognition, neural networks, data visualization, information retrieval, image and signal processing, and spatial data analysis. We adopt a database perspective in our presentation of data mining in this book. That is, emphasis is placed on *efficient* and *scalable* data mining techniques for *large* databases. By performing data mining, interesting knowledge, regularities, or high-level information can be extracted from databases and viewed or browsed from different angles. The discovered knowledge can be applied to decision making, process control, information management, query processing, and so on. Therefore, data mining is considered as one of the most important frontiers in database systems and one of the most promising, new database applications in the information industry.

1.3 Data mining — on what kind of data?

In this section, we examine a number of different data stores on which mining can be performed. In principle, data mining should be applicable to any kind of information repository. This includes relational databases, data warehouses, transactional databases, advanced database systems, flat files, and the World-Wide Web. Advanced database systems include object-oriented and object-relational databases, and specific application-oriented databases, such as spatial databases, time-series databases, text databases, and multimedia databases. The challenges and techniques of mining may differ for each of the repository systems.

Although this book assumes that readers have primitive knowledge of information systems, we provide a brief introduction to each of the major data repository systems listed above. In this section, we also introduce the fictitious *AllElectronics* store which will be used to illustrate concepts throughout the text.

1.3.1 Relational databases

A database system, also called a **database management system (DBMS)**, consists of a collection of interrelated data, known as a **database**, and a set of software programs to manage and access the data. The software programs involve mechanisms for the definition of database structures, for data storage, for concurrent, shared or distributed data access, and for ensuring the consistency and security of the information stored, despite system crashes or attempts at unauthorized access.

A **relational database** is a collection of **tables**, each of which is assigned a unique name. Each table consists of a set of **attributes** (*columns* or *fields*) and usually stores a large number of **tuples** (*records* or *rows*). Each tuple in a relational table represents an object identified by a unique *key* and described by a set of attribute values.

Consider the following example.

Example 1.1 The *AllElectronics* company is described by the following relation tables: *customer, item, employee*, and *branch.* Fragments of the tables described here are shown in Figure 1.6. The attribute which represents key or composite key component of each relation is underlined.

- The relation *customer* consists of a set of attributes, including a unique customer identity number (*cust_ID*), customer name, address, age, occupation, annual income, credit information, category, etc.

- Similarly, each of the relations *employee*, *branch*, and *items*, consists of a set of attributes, describing their properties.

- Tables can also be used to represent the relationships between or among multiple relation tables. For our example, these include *purchases* (customer purchases items, creating a sales transaction that is handled by an employee), *items_sold* (lists the items sold in a given transaction), and *works_at* (employee works at a branch of *AllElectronics*). □

Relational data can be accessed by **database queries** written in a relational query language, such as SQL, or with the assistance of graphical user interfaces. In the latter, the user may employ a menu, for example, to specify attributes to be included in the query, and the constraints on these attributes. A given query is transformed into a set of relational operations, such as join, selection, and projection, and is then optimized for efficient processing. A query allows retrieval of specified subsets of the data. Suppose that your job is to analyze the *AllElectronics* data. Through the use of relational queries, you can ask things like "Show me a list of all items that were sold in the last quarter". Relational languages also include aggregate functions such as `sum`, `avg` (average), `count`, `max` (maximum), and `min` (minimum). These allow you to find out things like "Show me the total sales of the last month, grouped by branch", or "How many sales transactions occurred in the month of December?", or "Which sales person had the highest amount of sales?".

When data mining is applied to relational databases, one can go further by *searching for trends or data patterns.* For example, data mining systems may analyze customer data to predict the credit risk of new customers based on their income, age, and previous credit information. Data mining systems may also detect deviations, such as items whose sales are far from those expected in comparison with the previous year. Such deviations can then be further investigated, e.g., has there been a change in packaging of such items, or a significant increase in price?

Relational databases are one of the most popularly available and rich information repositories for data mining, and thus they are a major data form in our study of data mining.

customer

cust_ID	name	address	age	income	credit_info	...
C1	Smith, Sandy	5463 E. Hastings, Burnaby, BC, V5A 4S9, Canada	21	$27000	1	...
...	...	...	...	...	...	...

item

item_ID	name	brand	category	type	price	place_made	supplier	cost
I3	hi-res-TV	Toshiba	high resolution	TV	$988.00	Japan	NikoX	$600.00
I8	multidisc-CDplay	Sanyo	multidisc	CD player	$369.00	Japan	MusicFront	$120.00
...	...	...	...	...	...	...	...	...

employee

empl_ID	name	category	group	salary	commission
E55	Jones, Jane	home entertainment	manager	$18,000	2%
...	...	...	...	...	...

branch

branch_ID	name	address
B1	City Square	369 Cambie St., Vancouver, BC V5L 3A2, Canada
...	...	...

purchases

trans_ID	cust_ID	empl_ID	date	time	method_paid	amount
T100	C1	E55	09/21/98	15:45	Visa	$1357.00
...	...	...	...	...	...	...

items_sold

trans_ID	item_ID	qty
T100	I3	1
T100	I8	2
...	...	...

works_at

empl_ID	branch_ID
E55	B1
...	...

Figure 1.6: Fragments of relations from a relational database for *AllElectronics*.

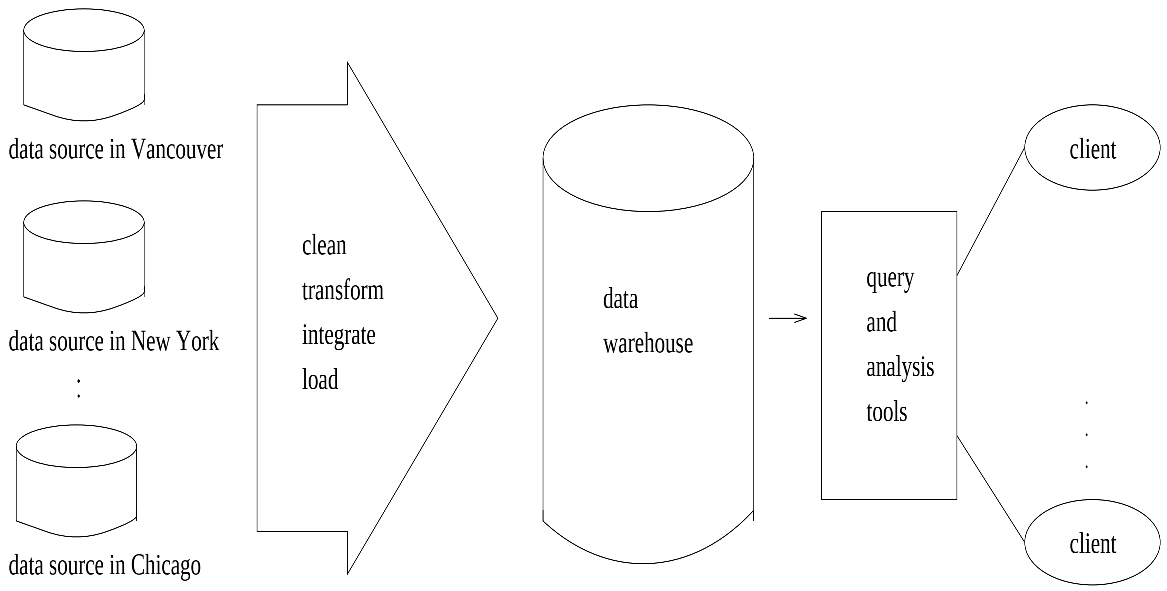


Figure 1.7: Architecture of a typical data warehouse.

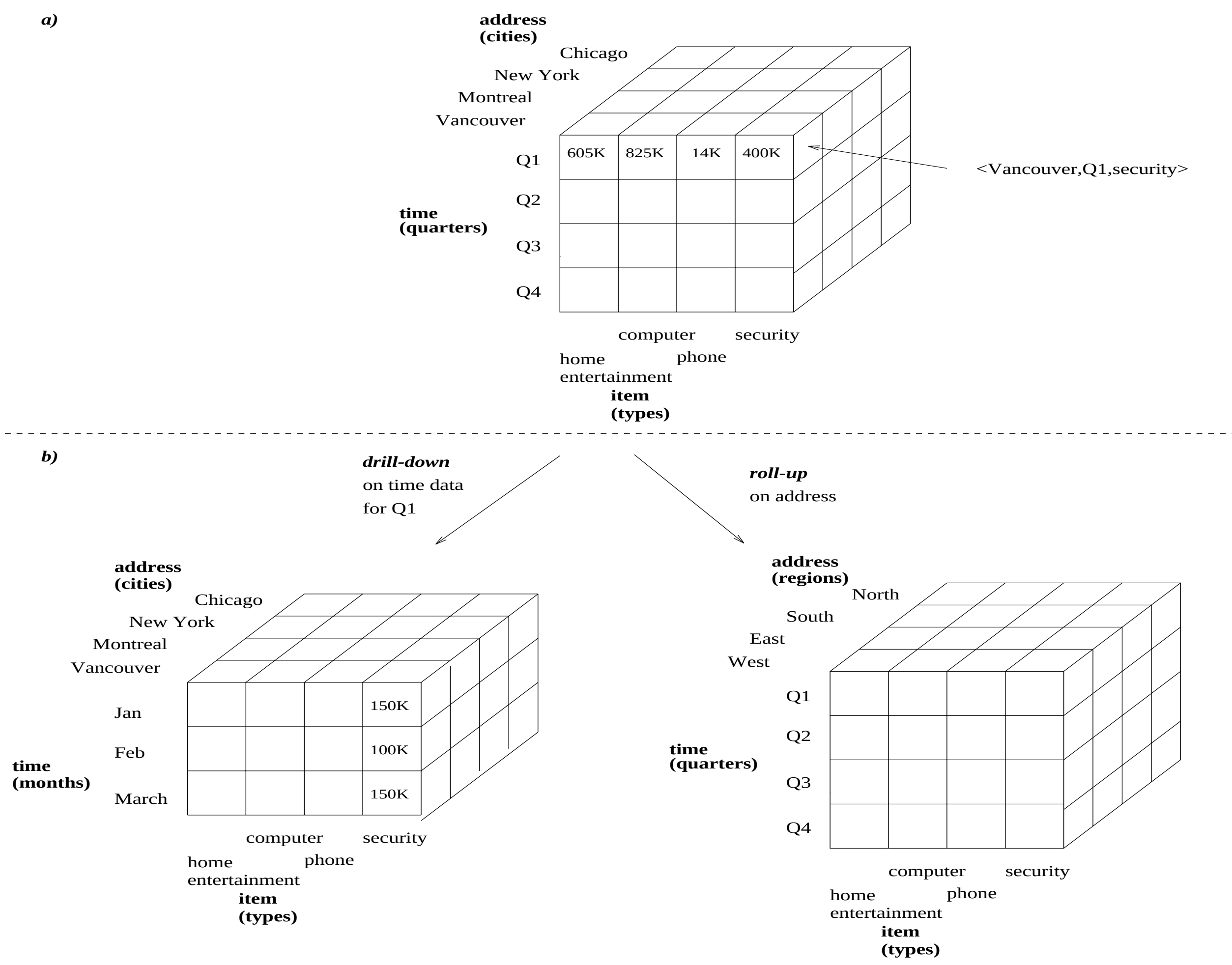


Figure 1.8: A multidimensional data cube, commonly used for data warehousing, *a)* showing summarized data for *AllElectronics* and *b)* showing summarized data resulting from drill-down and roll-up operations on the cube in *a)*.

1.3.2 Data warehouses

Suppose that *AllElectronics* is a successful international company, with branches around the world. Each branch has its own set of databases. The president of *AllElectronics* has asked you to provide an analysis of the company's sales per item type per branch for the third quarter. This is a difficult task, particularly since the relevant data are spread out over several databases, physically located at numerous sites.

If *AllElectronics* had a data warehouse, this task would be easy. A **data warehouse** is a repository of information collected from multiple sources, stored under a unified schema, and which usually resides at a single site. Data warehouses are constructed via a process of data cleansing, data transformation, data integration, data loading, and periodic data refreshing. This process is studied in detail in Chapter 2. Figure 1.7 shows the basic architecture of a data warehouse for *AllElectronics*.

In order to facilitate decision making, the data in a data warehouse are *organized around major subjects*, such as customer, item, supplier, and activity. The data are stored to provide information from a *historical perspective* (such as from the past 5-10 years), and are typically *summarized*. For example, rather than storing the details of each sales transaction, the data warehouse may store a summary of the transactions per item type for each store, or, summarized to a higher level, for each sales region.

A data warehouse is usually modeled by a multidimensional database structure, where each **dimension** corresponds to an attribute or a set of attributes in the schema, and each **cell** stores the value of some aggregate measure, such as *count* or *sales_amount*. The actual physical structure of a data warehouse may be a relational data store or a **multidimensional data cube**. It provides a multidimensional view of data and allows the precomputation and *sales*

trans_ID	list of item_ID's
T100	I1, I3, I8, I16
...	...

Figure 1.9: Fragment of a transactional database for sales at *AllElectronics*.

fast accessing of summarized data.

Example 1.2 A data cube for summarized sales data of *AllElectronics* is presented in Figure 1.8a). The cube has three **dimensions**: address (with *city* values *Chicago, New York, Montreal, Vancouver*), time (with *quarter* values *Q1, Q2, Q3, Q4*), and item (with item *type* values *home entertainment, computer, phone, security*). The aggregate value stored in each cell of the cube is *sales_amount*. For example, the total sales for $Q1$ of items relating to security systems in Vancouver is \$400K, as stored in cell ⟨Vancouver, Q1, security⟩. Additional cubes may be used to store aggregate sums over each dimension, corresponding to the aggregate values obtained using different SQL group-bys, e.g., the total sales amount per city and quarter, or per city and item, or per quarter and item, or per each individual dimension. □

In research literature on data warehouses, the data cube structure that stores the primitive or lowest level of information is called a **base cuboid**. Its corresponding higher level multidimensional (cube) structures are called (non-base) **cuboids**. A base cuboid together with all of its corresponding higher level cuboids form a **data cube**.

By providing multidimensional data views and the precomputation of summarized data, data warehouse systems are well suited for **On-Line Analytical Processing**, or **OLAP**. OLAP operations make use of background knowledge regarding the domain of the data being studied in order to allow the presentation of data at *different levels of abstraction*. Such operations accommodate different user viewpoints. Examples of OLAP operations include **drill-down** and **roll-up**, which allow the user to view the data at differing degrees of summarization, as illustrated in Figure 1.8b). For instance, one may drill down on sales data summarized by *quarter* to see the data summarized by *month*. Similarly, one may roll up on sales data summarized by *city* to view the data summarized by *region*.

Although data warehouse tools help support data analysis, additional tools for data mining are required to allow more in depth and automated analysis. Data warehouse technology is discussed in detail in Chapter 2.

1.3.3 Transactional databases

In general, a **transactional database** consists of a file where each record represents a transaction. A transaction typically includes a unique transaction identity number (*trans_ID*), and a list of the **items** making up the transaction (such as items purchased in a store). The transactional database may have additional tables associated with it, which contain other information regarding the sale, such as the date of the transaction, the customer ID number, the ID number of the sales person, and of the branch at which the sale occurred, and so on.

Example 1.3 Transactions can be stored in a table, with one record per transaction. A fragment of a transactional database for *AllElectronics* is shown in Figure 1.9. From the relational database point of view, the *sales* table in Figure 1.9 is a nested relation because the attribute "list of item_ID's" contains a set of *items*. Since most relational database systems do not support nested relational structures, the transactional database is usually either stored in a flat file in a format similar to that of the table in Figure 1.9, or unfolded into a standard relation in a format similar to that of the *items_sold* table in Figure 1.6. □

As an analyst of the *AllElectronics* database, you may like to ask "Show me all the items purchased by Sandy Smith" or "How many transactions include item number I3?". Answering such queries may require a scan of the entire transactional database.

Suppose you would like to dig deeper into the data by asking "Which items sold well together?". This kind of *market basket data analysis* would enable you to bundle groups of items together as a strategy for maximizing sales. For example, given the knowledge that printers are commonly purchased together with computers, you could offer an expensive model of printers at a discount to customers buying selected computers, in the hopes of selling more of the expensive printers. A regular data retrieval system is not able to answer queries like the one above. However, data mining systems for transactional data can do so by identifying sets of items which are frequently sold together.

1.3.4 Advanced database systems and advanced database applications

Relational database systems have been widely used in business applications. With the advances of database technology, various kinds of advanced database systems have emerged and are undergoing development to address the requirements of new database applications.

The new database applications include handling spatial data (such as maps), engineering design data (such as the design of buildings, system components, or integrated circuits), hypertext and multimedia data (including text, image, video, and audio data), time-related data (such as historical records or stock exchange data), and the World-Wide Web (a huge, widely distributed information repository made available by Internet). These applications require efficient data structures and scalable methods for handling complex object structures, variable length records, semi-structured or unstructured data, text and multimedia data, and database schemas with complex structures and dynamic changes.

In response to these needs, advanced database systems and specific application-oriented database systems have been developed. These include object-oriented and object-relational database systems, spatial database systems, temporal and time-series database systems, text and multimedia database systems, heterogeneous and legacy database systems, and the Web-based global information systems.

While such databases or information repositories require sophisticated facilities to efficiently store, retrieve, and update large amounts of complex data, they also provide fertile grounds and raise many challenging research and implementation issues for data mining.

1.4 Data mining functionalities — what kinds of patterns can be mined?

We have observed various types of data stores and database systems on which data mining can be performed. Let us now examine the kinds of data patterns that can be mined.

Data mining functionalities are used to specify the kind of patterns to be found in data mining tasks. In general, data mining tasks can be classified into two categories: **descriptive** and **predictive**. Descriptive mining tasks characterize the general properties of the data in the database. Predictive mining tasks perform inference on the current data in order to make predictions.

In some cases, users may have no idea of which kinds of patterns in their data may be interesting, and hence may like to search for several different kinds of patterns in parallel. Thus it is important to have a data mining system that can mine multiple kinds of patterns to accommodate different user expectations or applications. Furthermore, data mining systems should be able to discover patterns at various granularities (i.e., different levels of abstraction). To encourage interactive and exploratory mining, users should be able to easily "play" with the output patterns, such as by mouse clicking. Operations that can be specified by simple mouse clicks include adding or dropping a dimension (or an attribute), swapping rows and columns (**pivoting**, or axis rotation), changing dimension representations (e.g., from a 3-D cube to a sequence of 2-D cross tabulations, or **crosstabs**), or using OLAP roll-up or drill-down operations along dimensions. Such operations allow data patterns to be expressed from different angles of view and at multiple levels of abstraction.

Data mining systems should also allow users to specify hints to guide or focus the search for interesting patterns. Since some patterns may not hold for all of the data in the database, a measure of certainty or "trustworthiness" is usually associated with each discovered pattern.

Data mining functionalities, and the kinds of patterns they can discover, are described below.

1.4.1 Concept/class description: characterization and discrimination

Data can be associated with classes or concepts. For example, in the *AllElectronics* store, classes of items for sale include *computers* and *printers*, and concepts of customers include *bigSpenders* and *budgetSpenders*. It can be useful to describe individual classes and concepts in summarized, concise, and yet precise terms. Such descriptions of a class or a concept are called **class/concept descriptions**. These descriptions can be derived via (1) *data* *characterization*, by summarizing the data of the class under study (often called the **target class**) in general terms, or (2) *data discrimination*, by comparison of the target class with one or a set of comparative classes (often called the **contrasting classes**), or (3) both data characterization and discrimination.

Data **characterization** is a summarization of the general characteristics or features of a target class of data. The data corresponding to the user-specified class are typically collected by a database query. For example, to study the characteristics of software products whose sales increased by 10% in the last year, one can collect the data related to such products by executing an SQL query.

There are several methods for effective data summarization and characterization. For instance, the data cube-based OLAP roll-up operation (Section 1.3.2) can be used to perform user-controlled data summarization along a specified dimension. This process is further detailed in Chapter 2 which discusses data warehousing. An *attribute-oriented induction* technique can be used to perform data generalization and characterization without step-by-step user interaction. This technique is described in Chapter 5.

The output of data characterization can be presented in various forms. Examples include **pie charts**, **bar charts**, **curves**, **multidimensional data cubes**, and **multidimensional tables**, including crosstabs. The resulting descriptions can also be presented as **generalized relations**, or in rule form (called **characteristic rules**). These different output forms and their transformations are discussed in Chapter 5.

Example 1.4 A data mining system should be able to produce a description summarizing the characteristics of customers who spend more than $1000 a year at *AllElectronics*. The result could be a general profile of the customers such as they are 40-50 years old, employed, and have excellent credit ratings. The system should allow users to drill-down on any dimension, such as on "employment" in order to view these customers according to their occupation. □

Data **discrimination** is a comparison of the general features of target class data objects with the general features of objets from one or a set of contrasting classes. The target and contrasting classes can be specified by the user, and the corresponding data objects retrieved through data base queries. For example, one may like to compare the general features of software products whose sales increased by 10% in the last year with those whose sales decreased by at least 30% during the same period.

The methods used for data discrimination are similar to those used for data characterization. The forms of output presentation are also similar, although discrimination descriptions should include comparative measures which help distinguish between the target and contrasting classes. Discrimination descriptions expressed in rule form are referred to as **discriminant rules**. The user should be able to manipulate the output for characteristic and discriminant descriptions.

Example 1.5 A data mining system should be able to compare two groups of *AllElectronics* customers, such as those who shop for computer products regularly (more than 4 times a month) vs. those who rarely shop for such products (i.e., less than three times a year). The resulting description could be a general, comparative profile of the customers such as 80% of the customers who frequently purchase computer products are between 20-40 years old and have a university education, whereas 60% of the customers who infrequently buy such products are either old or young, and have no university degree. Drilling-down on a dimension, such as *occupation*, or adding new dimensions, such as *income_level*, may help in finding even more discriminative features between the two classes. □

Concept description, including characterization and discrimination, is the topic of Chapter 5.

1.4.2 Association analysis

Association analysis is the discovery of *association rules* showing attribute-value conditions that occur frequently together in a given set of data. Association analysis is widely used for market basket or transaction data analysis.

More formally, **association rules** are of the form $X \Rightarrow Y$, i.e., "$A_1 \wedge \cdots \wedge A_m \rightarrow B_1 \wedge \cdots \wedge B_n$", where A_i (for $i \in \{1, \ldots, m\}$) and B_j (for $j \in \{1, \ldots, n\}$) are attribute-value pairs. The association rule $X \Rightarrow Y$ is interpreted as "database tuples that satisfy the conditions in X are also likely to satisfy the conditions in Y".

Example 1.6 Given the *AllElectronics* relational database, a data mining system may find association rules like

$$age(X, \text{"}20-29\text{"}) \wedge income(X, \text{"}20-30K\text{"}) \Rightarrow buys(X, \text{"}CD\ player\text{"}) \qquad [support = 2\%, confidence = 60\%]$$

meaning that of the *AllElectronics* customers under study, 2% (**support**) are 20-29 years of age with an income of 20-30K and have purchased a CD player at *AllElectronics*. There is a 60% probability (**confidence**, or certainty) that a customer in this age and income group will purchase a CD player.

Note that this is an association between more than one attribute, or predicate (i.e., *age, income*, and *buys*). Adopting the terminology used in multidimensional databases, where each attribute is referred to as a dimension, the above rule can be referred to as a **multidimensional association rule**.

Suppose, as a marketing manager of *AllElectronics*, you would like to determine which items are frequently purchased together within the same transactions. An example of such a rule is

$$contains(T, \text{``}computer\text{''}) \ \Rightarrow\ contains(T, \text{``}software\text{''}) \qquad [support = 1\%, confidence = 50\%]$$

meaning that if a transaction T contains "computer", there is a 50% chance that it contains "software" as well, and 1% of all of the transactions contain both. This association rule involves a single attribute or predicate (i.e., *contains*) which repeats. Association rules that contain a single predicate are referred to as **single-dimensional association rules**. Dropping the predicate notation, the above rule can be written simply as "*computer* $\Rightarrow$ *software* [1%, 50%]". □

In recent years, many algorithms have been proposed for the efficient mining of association rules. Association rule mining is discussed in detail in Chapter 6.

1.4.3 Classification and prediction

Classification is the processing of finding a set of **models** (or functions) which describe and distinguish data classes or concepts, for the purposes of being able to use the model to predict the class of objects whose class label is unknown. The derived model is based on the analysis of a set of **training data** (i.e., data objects whose class label is known).

The derived model may be represented in various forms, such as *classification (IF-THEN) rules*, *decision trees*, *mathematical formulae*, or *neural networks*. A **decision tree** is a flow-chart-like tree structure, where each node denotes a test on an attribute value, each branch represents an outcome of the test, and tree leaves represent classes or class distributions. Decision trees can be easily converted to classification rules. A **neural network** is a collection of linear threshold units that can be trained to distinguish objects of different classes.

Classification can be used for predicting the class label of data objects. However, in many applications, one may like to predict some missing or unavailable *data values* rather than class labels. This is usually the case when the predicted values are numerical data, and is often specifically referred to as **prediction**. Although prediction may refer to both data value prediction and class label prediction, it is usually confined to data value prediction and thus is distinct from classification. Prediction also encompasses the identification of distribution *trends* based on the available data.

Classification and prediction may need to be preceded by **relevance analysis** which attempts to identify attributes that do not contribute to the classification or prediction process. These attributes can then be excluded.

Example 1.7 Suppose, as sales manager of *AllElectronics*, you would like to classify a large set of items in the store, based on three kinds of responses to a sales campaign: *good response*, *mild response*, and *no response*. You would like to derive a model for each of these three classes based on the descriptive features of the items, such as *price*, *brand, place_made, type*, and *category*. The resulting classification should maximally distinguish each class from the others, presenting an organized picture of the data set. Suppose that the resulting classification is expressed in the form of a decision tree. The decision tree, for instance, may identify *price* as being the single factor which best distinguishes the three classes. The tree may reveal that, after *price*, other features which help further distinguish objects of each class from another include *brand* and *place_made*. Such a decision tree may help you understand the impact of the given sales campaign, and design a more effective campaign for the future. □

Chapter 7 discusses classification and prediction in further detail.

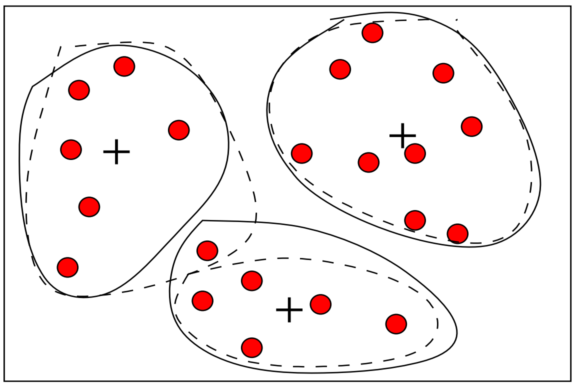


Figure 1.10: A 2-D plot of customer data with respect to customer locations in a city, showing three data clusters. Each cluster 'center' is marked with a '+'.

1.4.4 Clustering analysis

Unlike classification and predication, which analyze class-labeled data objects, **clustering** analyzes data objects without consulting a known class label. In general, the class labels are not present in the training data simply because they are not known to begin with. Clustering can be used to generate such labels. The objects are clustered or grouped based on the principle of *maximizing the intraclass similarity and minimizing the interclass similarity.* That is, clusters of objects are formed so that objects within a cluster have high similarity in comparison to one another, but are very dissimilar to objects in other clusters. Each cluster that is formed can be viewed as a class of objects, from which rules can be derived. Clustering can also facilitate **taxonomy formation**, that is, the organization of observations into a hierarchy of classes that group similar events together.

Example 1.8 Clustering analysis can be performed on *AllElectronics* customer data in order to identify homogeneous subpopulations of customers. These clusters may represent individual target groups for marketing. Figure 1.10 shows a 2-D plot of customers with respect to customer locations in a city. Three clusters of data points are evident. □

Clustering analysis forms the topic of Chapter 8.

1.4.5 Evolution and deviation analysis

Data **evolution analysis** describes and models regularities or trends for objects whose behavior changes over time. Although this may include characterization, discrimination, association, classification, or clustering of *time-related* data, distinct features of such an analysis include time-series data analysis, sequence or periodicity pattern matching, and similarity-based data analysis.

Example 1.9 Suppose that you have the major stock market (time-series) data of the last several years available from the New York Stock Exchange and you would like to invest in shares of high-tech industrial companies. A data mining study of stock exchange data may identify stock evolution regularities for overall stocks and for the stocks of particular companies. Such regularities may help predict future trends in stock market prices, contributing to your decision making regarding stock investments. □

In the analysis of time-related data, it is often desirable not only to model the general evolutionary trend of the data, but also to identify data deviations which occur over time. **Deviations** are differences between measured values and corresponding references such as previous values or normative values. A data mining system performing deviation analysis, upon the detection of a set of deviations, may do the following: describe the characteristics of the deviations, try to explain the reason behind them, and suggest actions to bring the deviated values back to their expected values.

Example 1.10 A decrease in *total sales* at *AllElectronics* for the last month, in comparison to that of the same month of the last year, is a deviation pattern. Having detected a significant deviation, a data mining system may go further and attempt to explain the detected pattern (e.g., did the company have more sales personnel last year in comparison to the same period this year?). □

Data evolution and deviation analysis are discussed in Chapter 9.

1.5 Are all of the patterns interesting?

A data mining system has the potential to generate thousands or even millions of patterns, or rules. *Are all of the patterns interesting?* Typically not — only a small fraction of the patterns potentially generated would actually be of interest to any given user.

This raises some serious questions for data mining: *What makes a pattern interesting? Can a data mining system generate all of the interesting patterns? Can a data mining system generate only the interesting patterns?*

To answer the first question, a pattern is **interesting** if (1) it is *easily understood* by humans, (2) *valid* on new or test data with some degree of *certainty*, (3) potentially *useful*, and (4) *novel*. A pattern is also interesting if it validates a hypothesis that the user *sought to confirm*. An interesting pattern represents **knowledge**.

Several **objective measures of pattern interestingness** exist. These are based on the structure of discovered patterns and the statistics underlying them. An objective measure for association rules of the form $X \Rightarrow Y$ is rule **support**, representing the percentage of data samples that the given rule satisfies. Another objective measure for association rules is **confidence**, which assesses the degree of certainty of the detected association. It is defined as the conditional probability that a pattern Y is true given that X is true. More formally, support and confidence are defined as

$$support(X \Rightarrow Y) = Prob\{X \cup Y\}.$$

$$confidence(X \Rightarrow Y) = Prob\{Y|X\}.$$

In general, each interestingness measure is associated with a threshold, which may be controlled by the user. For example, rules that do not satisfy a confidence threshold of say, 50%, can be considered uninteresting. Rules below the threshold likely reflect noise, exceptions, or minority cases, and are probably of less value.

Although objective measures help identify interesting patterns, they are insufficient unless combined with subjective measures that reflect the needs and interests of a particular user. For example, patterns describing the characteristics of customers who shop frequently at *AllElectronics* should interest the marketing manager, but may be of little interest to analysts studying the same database for patterns on employee performance. Furthermore, many patterns that are interesting by objective standards may represent common knowledge, and therefore, are actually uninteresting. **Subjective interestingness measures** are based on user beliefs in the data. These measures find patterns interesting if they are **unexpected** (contradicting a user belief) or offer strategic information on which the user can act. In the latter case, such patterns are referred to as **actionable**. Patterns that are expected can be interesting if they confirm a hypothesis that the user wished to validate, or resemble a user's hunch.

The second question, *"Can a data mining system generate* all *of the interesting patterns?"*, refers to the **completeness** of a data mining algorithm. It is unrealistic and inefficient for data mining systems to generate all of the possible patterns. Instead, a focused search which makes use of interestingness measures should be used to control pattern generation. This is often sufficient to ensure the completeness of the algorithm. Association rule mining is an example where the use of interestingness measures can ensure the completeness of mining. The methods involved are examined in detail in Chapter 6.

Finally, the third question, *"Can a data mining system generate* only *the interesting patterns?"*, is an optimization problem in data mining. It is highly desirable for data mining systems to generate only the interesting patterns. This would be much more efficient for users and data mining systems, since neither would have to search through the patterns generated in order to identify the truely interesting ones. Such optimization remains a challenging issue in data mining.

Measures of pattern interestingness are essential for the efficient discovery of patterns of value to the given user. Such measures can be used after the data mining step in order to rank the discovered patterns according to their interestingness, filtering out the uninteresting ones. More importantly, such measures can be used to guide and

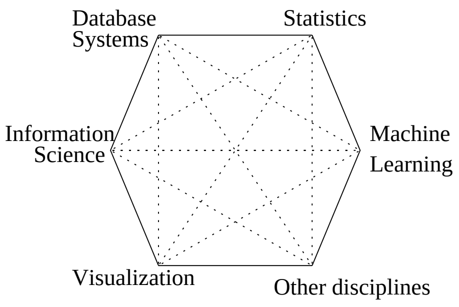


Figure 1.11: Data mining as a confluence of multiple disciplines.

constrain the discovery process, improving the search efficiency by pruning away subsets of the pattern space that do not satisfy pre-specified interestingness constraints.

Methods to assess pattern interestingness, and their use to improve data mining efficiency are discussed throughout the book, with respect to each kind of pattern that can be mined.

1.6 A classification of data mining systems

Data mining is an interdisciplinary field, the confluence of a set of disciplines (as shown in Figure 1.11), including database systems, statistics, machine learning, visualization, and information science. Moreover, depending on the data mining approach used, techniques from other disciplines may be applied, such as neural networks, fuzzy and/or rough set theory, knowledge representation, inductive logic programming, or high performance computing. Depending on the kinds of data to be mined or on the given data mining application, the data mining system may also integrate techniques from spatial data analysis, information retrieval, pattern recognition, image analysis, signal processing, computer graphics, Web technology, economics, or psychology.

Because of the diversity of disciplines contributing to data mining, data mining research is expected to generate a large variety of data mining systems. Therefore, it is necessary to provide a clear classification of data mining systems. Such a classification may help potential users distinguish data mining systems and identify those that best match their needs. Data mining systems can be categorized according to various criteria, as follows.

- Classification according to the *kinds of databases* mined.

  A data mining system can be classified according to the kinds of databases mined. Database systems themselves can be classified according to different criteria (such as data models, or the types of data or applications involved), each of which may require its own data mining technique. Data mining systems can therefore be classified accordingly.

  For instance, if classifying according to data models, we may have a relational, transactional, object-oriented, object-relational, or data warehouse mining system. If classifying according to the special types of data handled, we may have a spatial, time-series, text, or multimedia data mining system, or a World-Wide Web mining system. Other system types include heterogeneous data mining systems, and legacy data mining systems.

- Classification according to the *kinds of knowledge* mined.

  Data mining systems can be categorized according to the kinds of knowledge they mine, i.e., based on data mining functionalities, such as characterization, discrimination, association, classification, clustering, trend and evolution analysis, deviation analysis, similarity analysis, etc. A comprehensive data mining system usually provides multiple and/or integrated data mining functionalities.

  Moreover, data mining systems can also be distinguished based on the granularity or levels of abstraction of the knowledge mined, including generalized knowledge (at a high level of abstraction), primitive-level knowledge (at a raw data level), or knowledge at multiple levels (considering several levels of abstraction). An advanced data mining system should facilitate the discovery of knowledge at multiple levels of abstraction.

- Classification according to the *kinds of techniques* utilized.

Data mining systems can also be categorized according to the underlying data mining techniques employed. These techniques can be described according to the degree of user interaction involved (e.g., autonomous systems, interactive exploratory systems, query-driven systems), or the methods of data analysis employed (e.g., database-oriented or data warehouse-oriented techniques, machine learning, statistics, visualization, pattern recognition, neural networks, and so on). A sophisticated data mining system will often adopt multiple data mining techniques or work out an effective, integrated technique which combines the merits of a few individual approaches.

Chapters 5 to 8 of this book are organized according to the various kinds of knowledge mined. In Chapter 9, we discuss the mining of different kinds of data on a variety of advanced and application-oriented database systems.

1.7 Major issues in data mining

The scope of this book addresses major issues in data mining regarding mining methodology, user interaction, performance, and diverse data types. These issues are introduced below:

1. **Mining methodology and user-interaction issues.** These reflect the kinds of knowledge mined, the ability to mine knowledge at multiple granularities, the use of domain knowledge, ad-hoc mining, and knowledge visualization.

   - *Mining different kinds of knowledge in databases.*

     Since different users can be interested in different kinds of knowledge, data mining should cover a wide spectrum of data analysis and knowledge discovery tasks, including data characterization, discrimination, association, classification, clustering, trend and deviation analysis, and similarity analysis. These tasks may use the same database in different ways and require the development of numerous data mining techniques.

   - *Interactive mining of knowledge at multiple levels of abstraction.*

     Since it is difficult to know exactly what can be discovered within a database, the data mining process should be *interactive*. For databases containing a huge amount of data, appropriate sampling technique can first be applied to facilitate interactive data exploration. Interactive mining allows users to focus the search for patterns, providing and refining data mining requests based on returned results. Specifically, knowledge should be mined by drilling-down, rolling-up, and pivoting through the data space and knowledge space interactively, similar to what OLAP can do on data cubes. In this way, the user can interact with the data mining system to view data and discovered patterns at multiple granularities and from different angles.

   - *Incorporation of background knowledge.*

     Background knowledge, or information regarding the domain under study, may be used to guide the discovery process and allow discovered patterns to be expressed in concise terms and at different levels of abstraction. Domain knowledge related to databases, such as integrity constraints and deduction rules, can help focus and speed up a data mining process, or judge the interestingness of discovered patterns.

   - *Data mining query languages and ad-hoc data mining.*

     Relational query languages (such as SQL) allow users to pose ad-hoc queries for data retrieval. In a similar vein, high-level **data mining query languages** need to be developed to allow users to describe ad-hoc data mining tasks by facilitating the specification of the relevant sets of data for analysis, the domain knowledge, the kinds of knowledge to be mined, and the conditions and interestingness constraints to be enforced on the discovered patterns. Such a language should be integrated with a database or data warehouse query language, and optimized for efficient and flexible data mining.

   - *Presentation and visualization of data mining results.*

     Discovered knowledge should be expressed in high-level languages, visual representations, or other expressive forms so that the knowledge can be easily understood and directly usable by humans. This is especially crucial if the data mining system is to be interactive. This requires the system to adopt expressive knowledge representation techniques, such as trees, tables, rules, graphs, charts, crosstabs, matrices, or curves.

- *Handling outlier or incomplete data.*

  The data stored in a database may reflect outliers — noise, exceptional cases, or incomplete data objects. These objects may confuse the analysis process, causing overfitting of the data to the knowledge model constructed. As a result, the accuracy of the discovered patterns can be poor. Data cleaning methods and data analysis methods which can handle outliers are required. While most methods discard outlier data, such data may be of interest in itself such as in fraud detection for finding unusual usage of telecommunication services or credit cards. This form of data analysis is known as **outlier mining**.

- *Pattern evaluation: the interestingness problem.*

  A data mining system can uncover thousands of patterns. Many of the patterns discovered may be uninteresting to the given user, representing common knowledge or lacking novelty. Several challenges remain regarding the development of techniques to assess the interestingness of discovered patterns, particularly with regard to subjective measures which estimate the value of patterns with respect to a given user class, based on user beliefs or expectations. The use of interestingness measures to guide the discovery process and reduce the search space is another active area of research.

2. **Performance issues.** These include efficiency, scalability, and parallelization of data mining algorithms.

   - *Efficiency and scalability of data mining algorithms.*

     To effectively extract information from a huge amount of data in databases, data mining algorithms must be **efficient** and **scalable**. That is, the running time of a data mining algorithm must be predictable and acceptable in large databases. Algorithms with exponential or even medium-order polynomial complexity will not be of practical use. From a database perspective on knowledge discovery, efficiency and scalability are key issues in the implementation of data mining systems. Many of the issues discussed above under *mining methodology and user-interaction* must also consider efficiency and scalability.

   - *Parallel, distributed, and incremental updating algorithms.*

     The huge size of many databases, the wide distribution of data, and the computational complexity of some data mining methods are factors motivating the development of **parallel and distributed data mining algorithms**. Such algorithms divide the data into partitions, which are processed in parallel. The results from the partitions are then merged. Moreover, the high cost of some data mining processes promotes the need for **incremental** data mining algorithms which incorporate database updates without having to mine the entire data again "from scratch". Such algorithms perform knowledge modification incrementally to amend and strengthen what was previously discovered.

3. **Issues relating to the diversity of database types.**

   - *Handling of relational and complex types of data.*

     There are many kinds of data stored in databases and data warehouses. Can we expect that a single data mining system can perform effective mining on all kinds of data? Since relational databases and data warehouses are widely used, the development of efficient and effective data mining systems for such data is important. However, other databases may contain complex data objects, hypertext and multimedia data, spatial data, temporal data, or transaction data. It is unrealistic to expect one system to mine all kinds of data due to the diversity of data types and different goals of data mining. Specific data mining systems should be constructed for mining specific kinds of data. Therefore, one may expect to have different data mining systems for different kinds of data.

   - *Mining information from heterogeneous databases and global information systems.*

     Local and wide-area computer networks (such as the Internet) connect many sources of data, forming huge, distributed, and heterogeneous databases. The discovery of knowledge from different sources of structured, semi-structured, or unstructured data with diverse data semantics poses great challenges to data mining. Data mining may help disclose high-level data regularities in multiple heterogeneous databases that are unlikely to be discovered by simple query systems and may improve information exchange and interoperability in heterogeneous databases.

The above issues are considered major requirements and challenges for the further evolution of data mining technology. Some of the challenges have been addressed in recent data mining research and development, *to a* *certain extent*, and are now considered *requirements*, while others are still at the research stage. The issues, however, continue to stimulate further investigation and improvement. Additional issues relating to applications, privacy, and the social impact of data mining are discussed in Chapter 10, the final chapter of this book.

1.8 Summary

- **Database technology** has evolved from primitive file processing to the development of database management systems with query and transaction processing. Further progress has led to the increasing demand for efficient and effective data analysis and data understanding tools. This need is a result of the explosive growth in data collected from applications including business and management, government administration, scientific and engineering, and environmental control.
- **Data mining** is the task of discovering interesting patterns from large amounts of data where the data can be stored in databases, data warehouses, or other information repositories. It is a young interdisciplinary field, drawing from areas such as database systems, data warehousing, statistics, machine learning, data visualization, information retrieval, and high performance computing. Other contributing areas include neural networks, pattern recognition, spatial data analysis, image databases, signal processing, and inductive logic programming.
- A **knowledge discovery process** includes data cleaning, data integration, data selection, data transformation, data mining, pattern evaluation, and knowledge presentation.
- Data patterns can be mined from many different kinds of **databases**, such as relational databases, data warehouses, and transactional, object-relational, and object-oriented databases. Interesting data patterns can also be extracted from other kinds of **information repositories**, including spatial, time-related, text, multimedia, and legacy databases, and the World-Wide Web.
- A **data warehouse** is a repository for long term storage of data from multiple sources, organized so as to facilitate management decision making. The data are stored under a unified schema, and are typically summarized. Data warehouse systems provide some data analysis capabilities, collectively referred to as **OLAP (On-Line Analytical Processing)**. OLAP operations include drill-down, roll-up, and pivot.
- **Data mining functionalities** include the discovery of concept/class descriptions (i.e., characterization and discrimination), association, classification, prediction, clustering, trend analysis, deviation analysis, and similarity analysis. Characterization and discrimination are forms of data summarization.
- A pattern represents **knowledge** if it is easily understood by humans, valid on test data with some degree of certainty, potentially useful, novel, or validates a hunch about which the user was curious. Measures of **pattern interestingness**, either *objective* or *subjective*, can be used to guide the discovery process.
- **Data mining systems** can be **classified** according to the kinds of databases mined, the kinds of knowledge mined, or the techniques used.
- Efficient and effective data mining in large databases poses numerous **requirements** and great **challenges** to researchers and developers. The issues involved include data mining methodology, user-interaction, performance and scalability, and the processing of a large variety of data types. Other issues include the exploration of data mining applications, and their social impacts.

Exercises

1. What is data mining? In your answer, address the following:
   (a) Is it another hype?
   (b) Is it a simple transformation of technology developed from databases, statistics, and machine learning?
   (c) Explain how the evolution of database technology led to data mining.
   (d) Describe the steps involved in data mining when viewed as a process of knowledge discovery.

2. Present an example where data mining is crucial to the success of a business. What data mining functions does this business need? Can they be performed alternatively by data query processing or simple statistical analysis?

3. How is a data warehouse different from a database? How are they similar to each other?

4. Define each of the following data mining functionalities: characterization, discrimination, association, classification, prediction, clustering, and evolution and deviation analysis. Give examples of each data mining functionality, using a real-life database that you are familiar with.

5. Suppose your task as a software engineer at *Big-University* is to design a data mining system to examine their university course database, which contains the following information: the name, address, and status (e.g., undergraduate or graduate) of each student, and their cumulative grade point average (GPA). Describe the architecture you would choose. What is the purpose of each component of this architecture?

6. Based on your observation, describe another possible kind of knowledge that needs to be discovered by data mining methods but has not been listed in this chapter. Does it require a mining methodology that is quite different from those outlined in this chapter?

7. What is the difference between discrimination and classification? Between characterization and clustering? Between classification and prediction? For each of these pairs of tasks, how are they similar?

8. Describe three challenges to data mining regarding data mining methodology and user-interaction issues.

9. Describe two challenges to data mining regarding performance issues.

Bibliographic Notes

The book *Knowledge Discovery in Databases*, edited by Piatetsky-Shapiro and Frawley [26], is an early collection of research papers on knowledge discovery in databases. The book *Advances in Knowledge Discovery and Data Mining*, edited by Fayyad et al. [10], is a good collection of recent research results on knowledge discovery and data mining. Other books on data mining include *Predictive Data Mining* by Weiss and Indurkhya [37], and *Data Mining* by Adriaans and Zantinge [1]. There are also books containing collections of papers on particular aspects of knowledge discovery, such as *Machine Learning & Data Mining: Methods and Applications*, edited by Michalski, Bratko, and Kubat [20], *Rough Sets, Fuzzy Sets and Knowledge Discovery*, edited by Ziarko [39], as well as many tutorial notes on data mining, such as *Tutorial Notes of 1999 International Conference on Knowledge Disocvery and Data Mining* (*KDD99*) published by ACM Press.

KDD Nuggets is a regular, free electronic newsletter containing information relevant to knowledge discovery and data mining. Contributions can be e-mailed with a descriptive subject line (and a URL) to "gps@kdnuggets.com". Information regarding subscription can be found at "http://www.kdnuggets.com/subscribe.html". *KDD Nuggets* has been moderated by Piatetsky-Shapiro since 1991. The Internet site, *Knowledge Discovery Mine*, located at "http://www.kdnuggets.com/", contains a good collection of KDD-related information.

The research community of data mining set up a new academic organization called ACM-SIGKDD, a Special Interested Group on Knowledge Discovery in Databases under ACM in 1998. The community started its first international conference on knowledge discovery and data mining in 1995 [12]. The conference evolved from the four *international workshops on knowledge discovery in databases*, held from 1989 to 1994 [7, 8, 13, 11]. ACM-SIGKDD is organizing its first, but the fifth international conferences on knowledge discovery and data mining (KDD'99). A new journal, *Data Mining and Knowledge Discovery*, published by Kluwers Publishers, has been available since 1997.

Research in data mining has also been published in major textbooks, conferences and journals on databases, statistics, machine learning, and data visualization. References to such sources are listed below.

Popular textbooks on database systems include *Database System Concepts, 3rd ed.*, by Silberschatz, Korth, and Sudarshan [30], *Fundamentals of Database Systems, 2nd ed.*, by Elmasri and Navathe [9], and *Principles of Database and Knowledge-Base Systems, Vol. 1*, by Ullman [36]. For an edited collection of seminal articles on database systems, see *Readings in Database Systems* by Stonebraker [32]. Overviews and discussions on the achievements and research challenges in database systems can be found in Stonebraker et al. [33], and Silberschatz, Stonebraker, and Ullman [31].

Many books on data warehouse technology, systems and applications have been published in the last several years, such as *The Data Warehouse Toolkit* by Kimball [17], and *Building the Data Warehouse* by Inmon [14]. Chaudhuri and Dayal [3] present a comprehensive overview of data warehouse technology.

Research results relating to data mining and data warehousing have been published in the proceedings of many international database conferences, including *ACM-SIGMOD International Conference on Management of Data (SIGMOD), International Conference on Very Large Data Bases (VLDB), ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems (PODS), International Conference on Data Engineering (ICDE), International Conference on Extending Database Technology (EDBT), International Conference on Database Theory (ICDT), International Conference on Information and Knowledge Management (CIKM)*, and *International Symposium on Database Systems for Advanced Applications (DASFAA).* Research in data mining is also published in major database journals, such as *IEEE Transactions on Knowledge and Data Engineering (TKDE), ACM Transactions on Database Systems (TODS), Journal of ACM (JACM), Information Systems, The VLDB Journal, Data and Knowledge Engineering*, and *International Journal of Intelligent Information Systems (JIIS).*

There are many textbooks covering different topics in statistical analysis, such as *Probability and Statistics for Engineering and the Sciences, 4th ed.* by Devore [4], *Applied Linear Statistical Models, 4th ed.* by Neter et al. [25], *An Introduction to Generalized Linear Models* by Dobson [5], *Applied Statistical Time Series Analysis* by Shumway [29], and *Applied Multivariate Statistical Analysis, 3rd ed.* by Johnson and Wichern [15].

Research in statistics is published in the proceedings of several major statistical conferences, including *Joint Statistical Meetings*, *International Conference of the Royal Statistical Society*, and *Symposium on the Interface: Computing Science and Statistics.* Other source of publication include the *Journal of the Royal Statistical Society*, *The Annals of Statistics*, *Journal of American Statistical Association*, *Technometrics*, and *Biometrika.*

Textbooks and reference books on machine learning include *Machine Learning* by Mitchell [24], *Machine Learning, An Artificial Intelligence Approach*, Vols. 1-4, edited by Michalski et al. [21, 22, 18, 23], *C4.5: Programs for Machine Learning* by Quinlan [27], and *Elements of Machine Learning* by Langley [19]. The book *Computer Systems that Learn: Classification and Prediction Methods from Statistics, Neural Nets, Machine Learning, and Expert Systems*, by Weiss and Kulikowski [38], compares classification and prediction methods from several different fields, including statistics, machine learning, neural networks, and expert systems. For an edited collection of seminal articles on machine learning, see *Readings in Machine Learning* by Shavlik and Dietterich [28].

Machine learning research is published in the proceedings of several large machine learning and artificial intelligence conferences, including the *International Conference on Machine Learning (ML), ACM Conference on Computational Learning Theory (COLT), International Joint Conference on Artificial Intelligence (IJCAI)*, and *American Association of Artificial Intelligence Conference (AAAI).* Other sources of publication include major machine learning, artificial intelligence, and knowledge system journals, some of which have been mentioned above. Others include *Machine Learning (ML), Artificial Intelligence Journal (AI)* and *Cognitive Science.* An overview of classification from a statistical pattern recognition perspective can be found in Duda and Hart [6].

Pioneering work on data visualization techniques is described in *The Visual Display of Quantitative Information* [34] and *Envisioning Information* [35], both by Tufte, and *Graphics and Graphic Information Processing* by Bertin [2]. *Visual Techniques for Exploring Databases* by Keim [16] presents a broad tutorial on visualization for data mining. Major conferences and symposiums on visualization include *ACM Human Factors in Computing Systems (CHI)*, *Visualization*, and *International Symposium on Information Visualization.* Research on visualization is also published in *Transactions on Visualization and Computer Graphics*, *Journal of Computational and Graphical Statistics*, and *IEEE Computer Graphics and Applications.*

Bibliography

[1] P. Adriaans and D. Zantinge. *Data Mining.* Addison-Wesley: Harlow, England, 1996.

[2] J. Bertin. *Graphics and Graphic Information Processing.* Berlin, 1981.

[3] S. Chaudhuri and U. Dayal. An overview of data warehousing and OLAP technology. *ACM SIGMOD Record*, 26:65–74, 1997.

[4] J. L. Devore. *Probability and Statistics for Engineering and the Science, 4th ed.* Duxbury Press, 1995.

[5] A. J. Dobson. *An Introduction to Generalized Linear Models.* Chapman and Hall, 1990.

[6] R. Duda and P. Hart. *Pattern Classification and Scene Analysis.* Wiley: New York, 1973.

[7] G. Piatetsky-Shapiro (ed.). *Notes of IJCAI'89 Workshop Knowledge Discovery in Databases (KDD'89).* Detroit, Michigan, July 1989.

[8] G. Piatetsky-Shapiro (ed.). *Notes of AAAI'91 Workshop Knowledge Discovery in Databases (KDD'91).* Anaheim, CA, July 1991.

[9] R. Elmasri and S. B. Navathe. *Fundamentals of Database Systems, 2nd ed.* Bemjamin/Cummings, 1994.

[10] U. M. Fayyad, G. Piatetsky-Shapiro, P. Smyth, and R. Uthurusamy (eds.). *Advances in Knowledge Discovery and Data Mining.* AAAI/MIT Press, 1996.

[11] U.M. Fayyad and R. Uthurusamy (eds.). *Notes of AAAI'94 Workshop Knowledge Discovery in Databases (KDD'94).* Seattle, WA, July 1994.

[12] U.M. Fayyad and R. Uthurusamy (eds.). *Proc. 1st Int. Conf. Knowledge Discovery and Data Mining (KDD'95).* AAAI Press, Aug. 1995.

[13] U.M. Fayyad, R. Uthurusamy, and G. Piatetsky-Shapiro (eds.). *Notes of AAAI'93 Workshop Knowledge Discovery in Databases (KDD'93).* Washington, DC, July 1993.

[14] W. H. Inmon. *Building the Data Warehouse.* John Wiley, 1996.

[15] R. A. Johnson and D. W. Wickern. *Applied Multivariate Statistical Analysis, 3rd ed.* Prentice Hall, 1992.

[16] D. A. Keim. Visual techniques for exploring databases. In *Tutorial Notes, 3rd Int. Conf. on Knowledge Discovery and Data Mining (KDD'97)*, Newport Beach, CA, Aug. 1997.

[17] R. Kimball. *The Data Warehouse Toolkit.* John Wiley & Sons, New York, 1996.

[18] Y. Kodratoff and R. S. Michalski. *Machine Learning, An Artificial Intelligence Approach, Vol. 3.* Morgan Kaufmann, 1990.

[19] P. Langley. *Elements of Machine Learning.* Morgan Kaufmann, 1996.

[20] R. S. Michalski, I. Bratko, and M. Kubat. *Machine Learning and Data Mining: Methods and Applications.* John Wiley & Sons, 1998.

[21] R. S. Michalski, J. G. Carbonell, and T. M. Mitchell. *Machine Learning, An Artificial Intelligence Approach, Vol. 1.* Morgan Kaufmann, 1983.

[22] R. S. Michalski, J. G. Carbonell, and T. M. Mitchell. *Machine Learning, An Artificial Intelligence Approach, Vol. 2.* Morgan Kaufmann, 1986.

[23] R. S. Michalski and G. Tecuci. *Machine Learning, A Multistrategy Approach, Vol. 4.* Morgan Kaufmann, 1994.

[24] T. M. Mitchell. *Machine Learning.* McGraw Hill, 1997.

[25] J. Neter, M. H. Kutner, C. J. Nachtsheim, and L. Wasserman. *Applied Linear Statistical Models, 4th ed.* Irwin: Chicago, 1996.

[26] G. Piatetsky-Shapiro and W. J. Frawley. *Knowledge Discovery in Databases.* AAAI/MIT Press, 1991.

[27] J. R. Quinlan. *C4.5: Programs for Machine Learning.* Morgan Kaufmann, 1993.

[28] J.W. Shavlik and T.G. Dietterich. *Readings in Machine Learning.* Morgan Kaufmann, 1990.

[29] R. H. Shumway. *Applied Statistical Time Series Analysis.* Prentice Hall, 1988.

[30] A. Silberschatz, H. F. Korth, and S. Sudarshan. *Database System Concepts, 3ed.* McGraw-Hill, 1997.

[31] A. Silberschatz, M. Stonebraker, and J. D. Ullman. Database research: Achievements and opportunities into the 21st century. *ACM SIGMOD Record*, 25:52–63, March 1996.

[32] M. Stonebraker. *Readings in Database Systems, 2ed.* Morgan Kaufmann, 1993.

[33] M. Stonebraker, R. Agrawal, U. Dayal, E. Neuhold, and A. Reuter. DBMS research at a crossroads: The vienna update. In *Proc. 19th Int. Conf. Very Large Data Bases*, pages 688–692, Dublin, Ireland, Aug. 1993.

[34] E. R. Tufte. *The Visual Display of Quantitative Information.* Graphics Press, Cheshire, CT, 1983.

[35] E. R. Tufte. *Envisioning Information.* Graphics Press, Cheshire, CT, 1990.

[36] J. D. Ullman. *Principles of Database and Knowledge-Base Systems, Vol. 1.* Computer Science Press, 1988.

[37] S. M. Weiss and N. Indurkhya. *Predictive Data Mining.* Morgan Kaufmann, 1998.

[38] S. M. Weiss and C. A. Kulikowski. *Computer Systems that Learn: Classification and Prediction Methods from Statistics, Neural Nets, Machine Learning, and Expert Systems.* Morgan Kaufman, 1991.

[39] W. Ziarko. *Rough Sets, Fuzzy Sets and Knowledge Discovery.* Springer-Verlag, 1994.

Contents

Chapter 2

Data Warehouse and OLAP Technology for Data Mining

The construction of data warehouses, which involves data cleaning and data integration, can be viewed as an important preprocessing step for data mining. Moreover, data warehouses provide *on-line analytical processing (OLAP)* tools for the interactive analysis of multidimensional data of varied granularities, which facilitates effective data mining. Furthermore, many other data mining functions such as classification, prediction, association, and clustering, can be integrated with OLAP operations to enhance interactive mining of knowledge at multiple levels of abstraction. Hence, data warehouse has become an increasingly important platform for data analysis and on-line analytical processing and will provide an effective platform for data mining. Therefore, prior to presenting a systematic coverage of data mining technology in the remainder of this book, we devote this chapter to an overview of data warehouse technology. Such an overview is essential for understanding data mining technology.

In this chapter, you will learn the basic concepts, general architectures, and major implementation techniques employed in data warehouse and OLAP technology, as well as their relationship with data mining.

2.1 What is a data warehouse?

Data warehousing provides architectures and tools for business executives to systematically organize, understand, and use their data to make strategic decisions. A large number of organizations have found that data warehouse systems are valuable tools in today's competitive, fast evolving world. In the last several years, many firms have spent millions of dollars in building enterprise-wide data warehouses. Many people feel that with competition mounting in every industry, data warehousing is the latest must-have marketing weapon — a way to keep customers by learning more about their needs.

"So", you may ask, full of intrigue, *"what exactly is a data warehouse?"*

Data warehouses have been defined in many ways, making it difficult to formulate a rigorous definition. Loosely speaking, a data warehouse refers to a database that is maintained separately from an organization's operational databases. Data warehouse systems allow for the integration of a variety of application systems. They support information processing by providing a solid platform of consolidated, historical data for analysis.

According to W. H. Inmon, a leading architect in the construction of data warehouse systems, "a **data warehouse** is a **subject-oriented, integrated, time-variant**, and **nonvolatile** collection of data in support of management's decision making process." (Inmon 1992). This short, but comprehensive definition presents the major features of a data warehouse. The four keywords, *subject-oriented, integrated, time-variant*, and *nonvolatile*, distinguish data warehouses from other data repository systems, such as relational database systems, transaction processing systems, and file systems. Let's take a closer look at each of these key features.

- **Subject-oriented**: A data warehouse is organized around major subjects, such as customer, vendor, product, and sales. Rather than concentrating on the day-to-day operations and transaction processing of an organization, a data warehouse focuses on the modeling and analysis of data for decision makers. Hence, data warehouses typically provide a simple and concise view around particular subject issues by excluding data that are not useful in the decision support process.

- **Integrated**: A data warehouse is usually constructed by integrating multiple heterogeneous sources, such as relational databases, flat files, and on-line transaction records. Data cleaning and data integration techniques are applied to ensure consistency in naming conventions, encoding structures, attribute measures, and so on.
- **Time-variant**: Data are stored to provide information from a historical perspective (e.g., the past 5-10 years). Every key structure in the data warehouse contains, either implicitly or explicitly, an element of time.
- **Nonvolatile**: A data warehouse is always a physically separate store of data transformed from the application data found in the operational environment. Due to this separation, a data warehouse does not require transaction processing, recovery, and concurrency control mechanisms. It usually requires only two operations in data accessing: *initial loading of data* and *access of data.*

In sum, a data warehouse is a semantically consistent data store that serves as a physical implementation of a decision support data model and stores the information on which an enterprise needs to make strategic decisions. A data warehouse is also often viewed as an architecture, constructed by integrating data from multiple heterogeneous sources to support structured and/or ad hoc queries, analytical reporting, and decision making.

"OK", you now ask, *"what, then, is data warehousing?"*

Based on the above, we view **data warehousing** as the *process of constructing and using data warehouses.* The construction of a data warehouse requires data integration, data cleaning, and data consolidation. The utilization of a data warehouse often necessitates a collection of *decision support* technologies. This allows "knowledge workers" (e.g., managers, analysts, and executives) to use the warehouse to quickly and conveniently obtain an overview of the data, and to make sound decisions based on information in the warehouse. Some authors use the term "data warehousing" to refer only to the process of data warehouse *construction*, while the term **warehouse DBMS** is used to refer to the *management and utilization* of data warehouses. We will not make this distinction here.

"*How are organizations using the information from data warehouses?*" Many organizations are using this information to support business decision making activities, including (1) increasing customer focus, which includes the analysis of customer buying patterns (such as buying preference, buying time, budget cycles, and appetites for spending), (2) repositioning products and managing product portfolios by comparing the performance of sales by quarter, by year, and by geographic regions, in order to fine-tune production strategies, (3) analyzing operations and looking for sources of profit, and (4) managing the customer relationships, making environmental corrections, and managing the cost of corporate assets.

Data warehousing is also very useful from the point of view of *heterogeneous database integration.* Many organizations typically collect diverse kinds of data and maintain large databases from multiple, heterogeneous, autonomous, and distributed information sources. To integrate such data, and provide easy and efficient access to it is highly desirable, yet challenging. Much effort has been spent in the database industry and research community towards achieving this goal.

The traditional database approach to heterogeneous database integration is to build **wrappers** and **integrators** (or **mediators**) on top of multiple, heterogeneous databases. A variety of data joiner and data blade products belong to this category. When a query is posed to a client site, a metadata dictionary is used to translate the query into queries appropriate for the individual heterogeneous sites involved. These queries are then mapped and sent to local query processors. The results returned from the different sites are integrated into a global answer set. This **query-driven approach** requires complex information filtering and integration processes, and competes for resources with processing at local sources. It is inefficient and potentially expensive for frequent queries, especially for queries requiring aggregations.

Data warehousing provides an interesting alternative to the traditional approach of heterogeneous database integration described above. Rather than using a query-driven approach, data warehousing employs an **update-driven** approach in which information from multiple, heterogeneous sources is integrated in advance and stored in a warehouse for direct querying and analysis. Unlike on-line transaction processing databases, data warehouses do not contain the most current information. However, a data warehouse brings high performance to the integrated heterogeneous database system since data are copied, preprocessed, integrated, annotated, summarized, and restructured into one semantic data store. Furthermore, query processing in data warehouses does not interfere with the processing at local sources. Moreover, data warehouses can store and integrate historical information and support complex multidimensional queries. As a result, data warehousing has become very popular in industry.

Differences between operational database systems and data warehouses

Since most people are familiar with commercial relational database systems, it is easy to understand what a data warehouse is by comparing these two kinds of systems.

The major task of on-line operational database systems is to perform on-line transaction and query processing. These systems are called **on-line transaction processing (OLTP)** systems. They cover most of the day-to-day operations of an organization, such as, purchasing, inventory, manufacturing, banking, payroll, registration, and accounting. Data warehouse systems, on the other hand, serve users or "knowledge workers" in the role of data analysis and decision making. Such systems can organize and present data in various formats in order to accommodate the diverse needs of the different users. These systems are known as **on-line analytical processing (OLAP)** systems.

The major distinguishing features between OLTP and OLAP are summarized as follows.

1. **Users and system orientation**: An OLTP system is *customer-oriented* and is used for transaction and query processing by clerks, clients, and information technology professionals. An OLAP system is *market-oriented* and is used for data analysis by knowledge workers, including managers, executives, and analysts.

2. **Data contents**: An OLTP system manages current data that, typically, are too detailed to be easily used for decision making. An OLAP system manages large amounts of historical data, provides facilities for summarization and aggregation, and stores and manages information at different levels of granularity. These features make the data easier for use in informed decision making.

3. **Database design**: An OLTP system usually adopts an entity-relationship (ER) data model and an application-oriented database design. An OLAP system typically adopts either a *star* or *snowflake* model (to be discussed in Section 2.2.2), and a subject-oriented database design.

4. **View**: An OLTP system focuses mainly on the current data within an enterprise or department, without referring to historical data or data in different organizations. In contrast, an OLAP system often spans multiple versions of a database schema, due to the evolutionary process of an organization. OLAP systems also deal with information that originates from different organizations, integrating information from many data stores. Because of their huge volume, OLAP data are stored on multiple storage media.

5. **Access patterns**: The access patterns of an OLTP system consist mainly of short, atomic transactions. Such a system requires concurrency control and recovery mechanisms. However, accesses to OLAP systems are mostly read-only operations (since most data warehouses store historical rather than up-to-date information), although many could be complex queries.

Other features which distinguish between OLTP and OLAP systems include database size, frequency of operations, and performance metrics. These are summarized in Table 2.1.

But, why have a separate data warehouse?

"Since operational databases store huge amounts of data", you observe, *"why not perform on-line analytical processing directly on such databases instead of spending additional time and resources to construct a separate data warehouse?"*

A major reason for such a separation is to help promote the *high performance of both systems*. An operational database is designed and tuned from known tasks and workloads, such as indexing and hashing using primary keys, searching for particular records, and optimizing "canned" queries. On the other hand, data warehouse queries are often complex. They involve the computation of large groups of data at summarized levels, and may require the use of special data organization, access, and implementation methods based on multidimensional views. Processing OLAP queries in operational databases would substantially degrade the performance of operational tasks.

Moreover, an operational database supports the concurrent processing of several transactions. Concurrency control and recovery mechanisms, such as locking and logging, are required to ensure the consistency and robustness of transactions. An OLAP query often needs read-only access of data records for summarization and aggregation. Concurrency control and recovery mechanisms, if applied for such OLAP operations, may jeopardize the execution of concurrent transactions and thus substantially reduce the throughput of an OLTP system.

Feature	OLTP	OLAP
Characteristic	operational processing	informational processing
Orientation	transaction	analysis
User	clerk, DBA, database professional	knowledge worker (e.g., manager, executive, analyst)
Function	day-to-day operations	long term informational requirements, decision support
DB design	E-R based, application-oriented	star/snowflake, subject-oriented
Data	current; guaranteed up-to-date	historical; accuracy maintained over time
Summarization	primitive, highly detailed	summarized, consolidated
View	detailed, flat relational	summarized, multidimensional
Unit of work	short, simple transaction	complex query
Access	read/write	mostly read
Focus	data in	information out
Operations	index/hash on primary key	lots of scans
# of records accessed	tens	millions
# of users	thousands	hundreds
DB size	100 MB to GB	100 GB to TB
Priority	high performance, high availability	high flexibility, end-user autonomy
Metric	transaction throughput	query throughput, response time

Table 2.1: Comparison between OLTP and OLAP systems.

Finally, the separation of operational databases from data warehouses is based on the different structures, contents, and uses of the data in these two systems. Decision support requires historical data, whereas operational databases do not typically maintain historical data. In this context, the data in operational databases, though abundant, is usually far from complete for decision making. Decision support requires consolidation (such as aggregation and summarization) of data from heterogeneous sources, resulting in high quality, cleansed and integrated data. In contrast, operational databases contain only detailed raw data, such as transactions, which need to be consolidated before analysis. Since the two systems provide quite different functionalities and require different kinds of data, it is necessary to maintain separate databases.

2.2 A multidimensional data model

Data warehouses and OLAP tools are based on a **multidimensional data model**. This model views data in the form of a *data cube*. In this section, you will learn how data cubes model n-dimensional data. You will also learn about concept hierarchies and how they can be used in basic OLAP operations to allow interactive mining at multiple levels of abstraction.

2.2.1 From tables to data cubes

"What is a data cube?"

A **data cube** allows data to be modeled and viewed in multiple dimensions. It is defined by dimensions and facts.

In general terms, **dimensions** are the perspectives or entities with respect to which an organization wants to keep records. For example, *AllElectronics* may create a *sales* data warehouse in order to keep records of the store's sales with respect to the dimensions *time*, *item*, *branch*, and *location*. These dimensions allow the store to keep track of things like monthly sales of items, and the branches and locations at which the items were sold. Each dimension may have a table associated with it, called a **dimension table**, which further describes the dimension. For example, a dimension table for *item* may contain the attributes *item_name, brand*, and *type*. Dimension tables can be specified by users or experts, or automatically generated and adjusted based on data distributions.

A multidimensional data model is typically organized around a central theme, like *sales*, for instance. This theme is represented by a fact table. **Facts** are numerical measures. Think of them as the quantities by which we want to analyze relationships between dimensions. Examples of facts for a sales data warehouse include *dollars_sold* (sales amount in dollars), *units_sold* (number of units sold), and *amount_budgeted*. The **fact table** contains the names of the *facts*, or measures, as well as keys to each of the related dimension tables. You will soon get a clearer picture of how this works when we later look at multidimensional schemas.

Although we usually think of cubes as 3-D geometric structures, in data warehousing the data cube is n-dimensional. To gain a better understanding of data cubes and the multidimensional data model, let's start by looking at a simple 2-D data cube which is, in fact, a table for sales data from *AllElectronics*. In particular, we will look at the *AllElectronics* sales data for items sold per quarter in the city of Vancouver. These data are shown in Table 2.2. In this 2-D representation, the sales for Vancouver are shown with respect to the *time* dimension (organized in quarters) and the *item* dimension (organized according to the types of items sold). The fact, or measure displayed is *dollars_sold*.

time (quarter)	Sales for all **locations** in Vancouver			
	item (type)			
	home entertainment	computer	phone	security
Q1	605K	825K	14K	400K
Q2	680K	952K	31K	512K
Q3	812K	1023K	30K	501K
Q4	927K	1038K	38K	580K

Table 2.2: A 2-D view of sales data for *AllElectronics* according to the dimensions *time* and *item*, where the sales are from branches located in the city of Vancouver. The measure displayed is *dollars_sold*.

	location = "Vancouver"				**location** = "Montreal"				**location** = "New York"				**location** = "Chicago"			
time	**item**				**item**				**item**				**item**			
	home ent.	comp.	phone	sec.	home ent.	comp.	phone	sec.	home ent.	comp.	phone	sec.	home ent.	comp.	phone	sec.
Q1	605K	825K	14K	400K	818K	746K	43K	591K	1087K	968K	38K	872K	854K	882K	89K	623K
Q2	680K	952K	31K	512K	894K	769K	52K	682K	1130K	1024K	41K	925K	943K	890K	64K	698K
Q3	812K	1023K	30K	501K	940K	795K	58K	728K	1034K	1048K	45K	1002K	1032K	924K	59K	789K
Q4	927K	1038K	38K	580K	978K	864K	59K	784K	1142K	1091K	54K	984K	1129K	992K	63K	870K

Table 2.3: A 3-D view of sales data for *AllElectronics*, according to the dimensions *time*, *item*, and *location*. The measure displayed is *dollars_sold*.

Now, suppose that we would like to view the sales data with a third dimension. For instance, suppose we would like to view the data according to *time*, *item*, as well as *location*. These 3-D data are shown in Table 2.3. The 3-D data of Table 2.3 are represented as a series of 2-D tables. Conceptually, we may also represent the same data in the form of a *3-D* data cube, as in Figure 2.1.

Suppose that we would now like to view our sales data with an additional fourth dimension, such as *supplier*. Viewing things in 4-D becomes tricky. However, we can think of a 4-D cube as being a series of 3-D cubes, as shown in Figure 2.2. If we continue in this way, we may display any n-D data as a series of $(n-1)$-D "cubes". The data cube is a metaphor for multidimensional data storage. The actual physical storage of such data may differ from its logical representation. The important thing to remember is that data cubes are n-dimensional, and do not confine data to 3-D.

The above tables show the data at different degrees of summarization. In the data warehousing research literature, a data cube such as each of the above is referred to as a **cuboid**. Given a set of dimensions, we can construct a *lattice* of cuboids, each showing the data at a different level of summarization, or group by (i.e., summarized by a different subset of the dimensions). The lattice of cuboids is then referred to as a **data cube**. Figure 2.8 shows a lattice of cuboids forming a data cube for the dimensions *time*, *item*, *location*, and *supplier*.

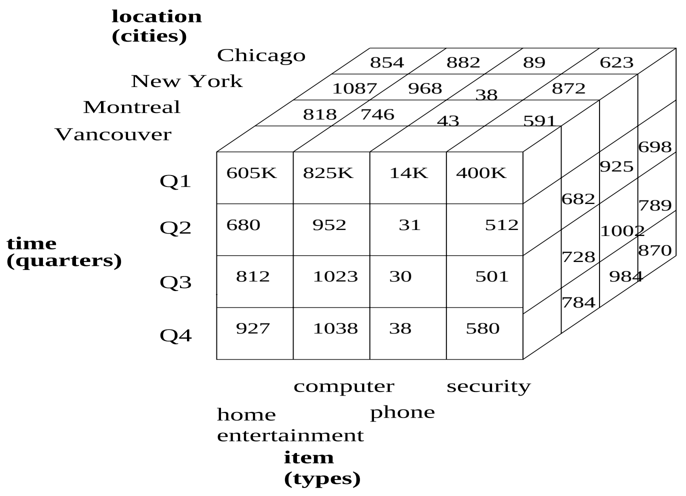


Figure 2.1: A 3-D data cube representation of the data in Table 2.3, according to the dimensions *time*, *item*, and *location*. The measure displayed is *dollars_sold*.

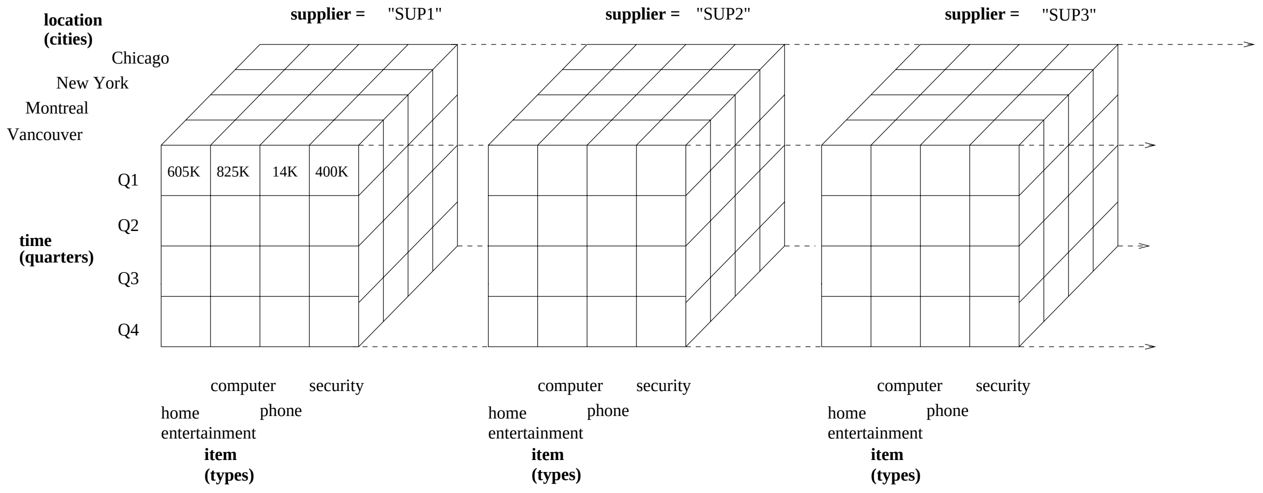


Figure 2.2: A 4-D data cube representation of sales data, according to the dimensions *time*, *item*, *location*, and *supplier*. The measure displayed is *dollars_sold*.

The cuboid which holds the lowest level of summarization is called the **base cuboid**. For example, the 4-D cuboid in Figure 2.2 is the base cuboid for the given *time*, *item*, *location*, and *supplier* dimensions. Figure 2.1 is a 3-D (non-base) cuboid for *time*, *item*, and *location*, summarized for all suppliers. The 0-D cuboid which holds the highest level of summarization is called the **apex cuboid**. In our example, this is the total sales, or *dollars_sold*, summarized for all four dimensions. The apex cuboid is typically denoted by all.

2.2.2 Stars, snowflakes, and fact constellations: schemas for multidimensional databases

The entity-relationship data model is commonly used in the design of relational databases, where a database schema consists of a set of entities or objects, and the relationships between them. Such a data model is appropriate for on-line transaction processing. Data warehouses, however, require a concise, subject-oriented schema which facilitates on-line data analysis.

The most popular data model for data warehouses is a **multidimensional model**. This model can exist in the form of a **star schema**, a **snowflake schema**, or a **fact constellation schema**. Let's have a look at each of these schema types.

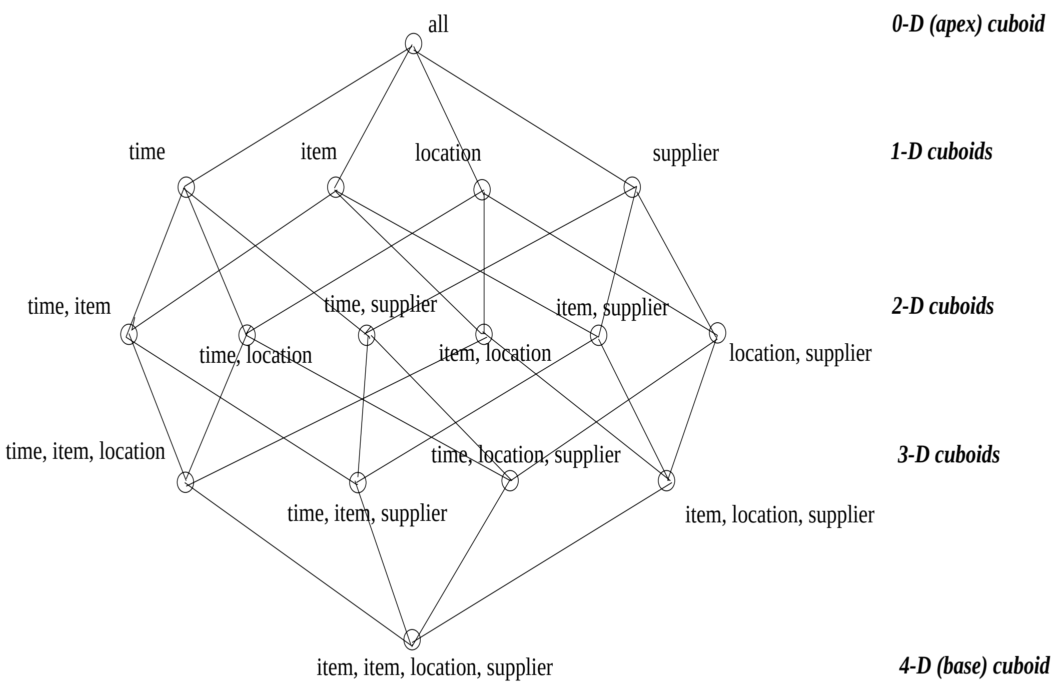


Figure 2.3: Lattice of cuboids, making up a 4-D data cube for the dimensions *time*, *item*, *location*, and *supplier*. Each cuboid represents a different degree of summarization.

- **Star schema**: The star schema is a modeling paradigm in which the data warehouse contains (1) a large central table (**fact table**), and (2) a set of smaller attendant tables (**dimension tables**), one for each dimension. The schema graph resembles a starburst, with the dimension tables displayed in a radial pattern around the central fact table.

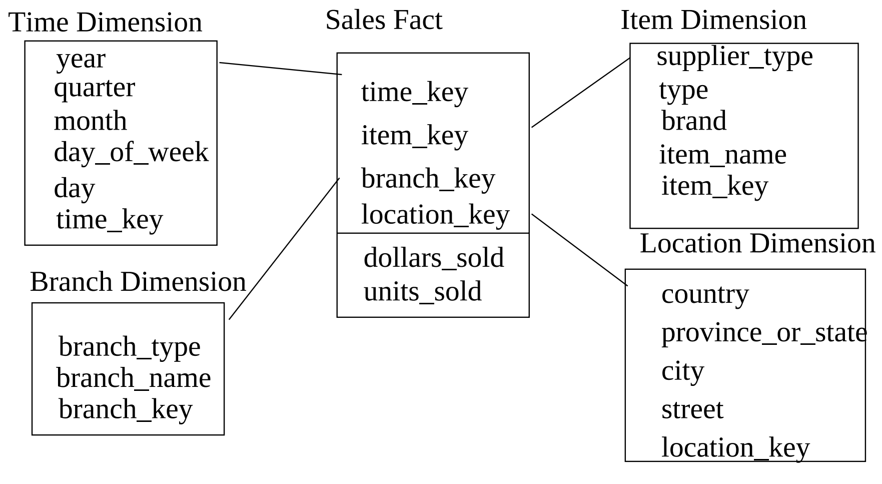


Figure 2.4: Star schema of a data warehouse for sales.

Example 2.1 An example of a star schema for *AllElectronics* sales is shown in Figure 2.4. Sales are considered along four dimensions, namely *time, item, branch*, and *location*. The schema contains a central fact table for *sales* which contains keys to each of the four dimensions, along with two measures: *dollars_sold* and *units_sold*. □

Notice that in the star schema, each dimension is represented by only one table, and each table contains a set of attributes. For example, the *location* dimension table contains the attribute set {*location_key, street,* *city, province_or_state, country*}. This constraint may introduce some redundancy. For example, "Vancouver" and "Victoria" are both cities in the Canadian province of British Columbia. Entries for such cities in the location dimension table will create redundancy among the attributes *province_or_state* and *country*, i.e., `(.., Vancouver, British Columbia, Canada)` and `(.., Victoria, British Columbia, Canada)`. Moreover, the attributes within a dimension table may form either a hierarchy (total order) or a lattice (partial order).

- **Snowflake schema**: The snowflake schema is a variant of the star schema model, where some dimension tables are *normalized*, thereby further splitting the data into additional tables. The resulting schema graph forms a shape similar to a snowflake.

  The major difference between the snowflake and star schema models is that the dimension tables of the snowflake model may be kept in normalized form. Such a table is easy to maintain and also saves storage space because a large dimension table can be extremely large when the dimensional structure is included as columns. Since much of this space is redundant data, creating a normalized structure will reduce the overall space requirement. However, the snowflake structure can reduce the effectiveness of browsing since more joins will be needed to execute a query. Consequently, the system performance may be adversely impacted. Performance benchmarking can be used to determine what is best for your design.

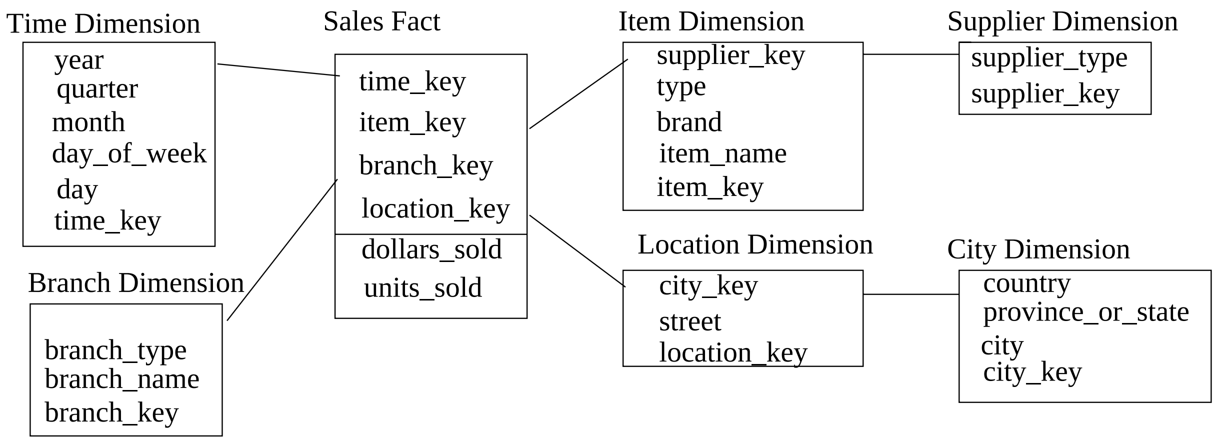


Figure 2.5: Snowflake schema of a data warehouse for sales.

Example 2.2 An example of a snowflake schema for *AllElectronics* sales is given in Figure 2.5. Here, the *sales* fact table is identical to that of the star schema in Figure 2.4. The main difference between the two schemas is in the definition of dimension tables. The single dimension table for *item* in the star schema is normalized in the snowflake schema, resulting in new *item* and *supplier* tables. For example, the *item* dimension table now contains the attributes *supplier_key*, *type*, *brand, item_name*, and *item_key*, the latter of which is linked to the *supplier* dimension table, containing *supplier_type* and *supplier_key* information. Similarly, the single dimension table for *location* in the star schema can be normalized into two tables: new *location* and *city*. The *location_key* of the new *location* table now links to the *city* dimension. Notice that further normalization can be performed on *province_or_state* and *country* in the snowflake schema shown in Figure 2.5, when desirable. □

A compromise between the star schema and the snowflake schema is to adopt a **mixed schema** where only the very large dimension tables are normalized. Normalizing large dimension tables saves storage space, while keeping small dimension tables unnormalized may reduce the cost and performance degradation due to joins on multiple dimension tables. Doing both may lead to an overall performance gain. However, careful performance tuning could be required to determine which dimension tables should be normalized and split into multiple tables.

- **Fact constellation**: Sophisticated applications may require multiple fact tables to *share* dimension tables. This kind of schema can be viewed as a collection of stars, and hence is called a **galaxy schema** or a **fact constellation**.

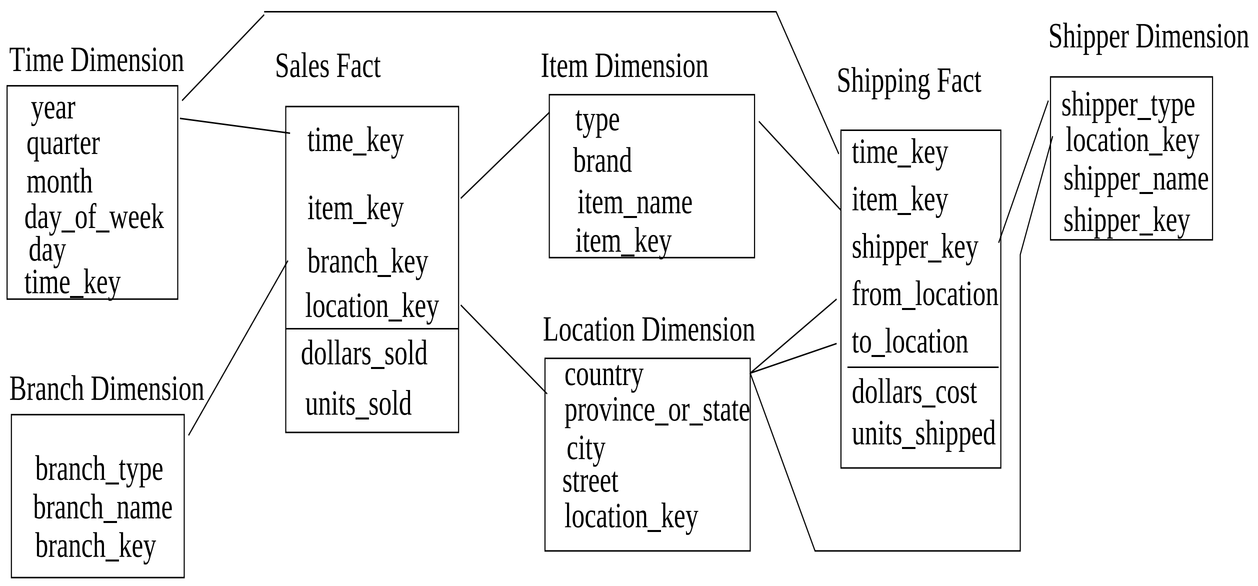


Figure 2.6: Fact constellation schema of a data warehouse for sales and shipping.

Example 2.3 An example of a fact constellation schema is shown in Figure 2.6. This schema specifies two fact tables, *sales* and *shipping*. The *sales* table definition is identical to that of the star schema (Figure 2.4). The *shipping* table has five dimensions, or keys: *time_key, item_key, shipper_key, from_location*, and *to_location*, and two measures: *dollars_cost* and *units_shipped*. A fact constellation schema allows dimension tables to be shared between fact tables. For example, the dimensions tables for *time, item*, and *location*, are shared between both the *sales* and *shipping* fact tables. □

In data warehousing, there is a distinction between a **data warehouse** and a **data mart**. A **data warehouse** collects information about subjects that span the *entire organization*, such as *customers, items, sales, assets*, and *personnel*, and thus its scope is *enterprise-wide*. For data warehouses, the fact constellation schema is commonly used since it can model multiple, interrelated subjects. A **data mart**, on the other hand, is a department subset of the data warehouse that focuses on selected subjects, and thus its scope is *department-wide*. For data marts, the *star* or *snowflake* schema are popular since each are geared towards modeling single subjects.

2.2.3 Examples for defining star, snowflake, and fact constellation schemas

"How can I define a multidimensional schema for my data?"

Just as relational query languages like SQL can be used to specify relational queries, a **data mining query language** can be used to specify data mining tasks. In particular, we examine an SQL-based data mining query language called **DMQL** which contains language primitives for defining data warehouses and data marts. Language primitives for specifying other data mining tasks, such as the mining of concept/class descriptions, associations, classifications, and so on, will be introduced in Chapter 4.

Data warehouses and data marts can be defined using two language primitives, one for *cube definition* and one for *dimension definition*. The *cube definition* statement has the following syntax.

define cube ⟨cube_name⟩ [⟨dimension_list⟩] : ⟨measure_list⟩

The *dimension definition* statement has the following syntax.

define dimension ⟨dimension_name⟩ as (⟨attribute_or_subdimension_list⟩)

Let's look at examples of how to define the star, snowflake and constellations schemas of Examples 2.1 to 2.3 using DMQL. DMQL keywords are displayed in sans serif font.

Example 2.4 The star schema of Example 2.1 and Figure 2.4 is defined in DMQL as follows.

```
define cube sales_star [time, item, branch, location]:
                         dollars_sold = sum(sales_in_dollars), units_sold = count(*)
define dimension time as (time_key, day, day_of_week, month, quarter, year)
define dimension item as (item_key, item_name, brand, type, supplier_type)
define dimension branch as (branch_key, branch_name, branch_type)
define dimension location as (location_key, street, city, province_or_state, country)
```

The define cube statement defines a data cube called *sales_star*, which corresponds to the central *sales* fact table of Example 2.1. This command specifies the keys to the dimension tables, and the two measures, *dollars_sold* and *units_sold*. The data cube has four dimensions, namely *time, item, branch*, and *location*. A define dimension statement is used to define each of the dimensions. □

Example 2.5 The snowflake schema of Example 2.2 and Figure 2.5 is defined in DMQL as follows.

```
define cube sales_snowflake [time, item, branch, location]:
                         dollars_sold = sum(sales_in_dollars), units_sold = count(*)
define dimension time as (time_key, day, day_of_week, month, quarter, year)
define dimension item as (item_key, item_name, brand, type, supplier (supplier_key, supplier_type))
define dimension branch as (branch_key, branch_name, branch_type)
define dimension location as (location_key, street, city (city_key, city, province_or_state, country))
```

This definition is similar to that of *sales_star* (Example 2.4), except that, here, the *item* and *location* dimensions tables are normalized. For instance, the *item* dimension of the *sales_star* data cube has been normalized in the *sales_snowflake* cube into two dimension tables, *item* and *supplier*. Note that the dimension definition for *supplier* is specified within the definition for *item*. Defining *supplier* in this way implicitly creates a *supplier_key* in the *item* dimension table definition. Similarly, the *location* dimension of the *sales_star* data cube has been normalized in the *sales_snowflake* cube into two dimension tables, *location* and *city*. The dimension definition for *city* is specified within the definition for *location*. In this way, a *city_key* is implicitly created in the *location* dimension table definition. □

Finally, a fact constellation schema can be defined as a set of interconnected cubes. Below is an example.

Example 2.6 The fact constellation schema of Example 2.3 and Figure 2.6 is defined in DMQL as follows.

```
define cube sales [time, item, branch, location]:
                         dollars_sold = sum(sales_in_dollars), units_sold = count(*)
define dimension time as (time_key, day, day_of_week, month, quarter, year)
define dimension item as (item_key, item_name, brand, type)
define dimension branch as (branch_key, branch_name, branch_type)
define dimension location as (location_key, street, city, province_or_state, country)

define cube shipping [time, item, shipper, from_location, to_location]:
                         dollars_cost = sum(cost_in_dollars), units_shipped = count(*)
define dimension time as time in cube sales
define dimension item as item in cube sales
define dimension shipper as (shipper_key, shipper_name, location as location in cube sales, shipper_type)
define dimension from_location as location in cube sales
define dimension to_location as location in cube sales
```

A define cube statement is used to define data cubes for *sales* and *shipping*, corresponding to the two fact tables of the schema of Example 2.3. Note that the *time, item*, and *location* dimensions of the *sales* cube are shared with the *shipping* cube. This is indicated for the *time* dimension, for example, as follows. Under the define cube statement for *shipping*, the statement "define dimension *time* as *time* in cube *sales*" is specified. □

Instead of having users or experts explicitly define data cube dimensions, dimensions can be automatically generated or adjusted based on the examination of data distributions. DMQL primitives for specifying such automatic generation or adjustments are discussed in the following chapter.

2.2.4 Measures: their categorization and computation

"How are measures computed?"

To answer this question, we will first look at how measures can be categorized. Note that multidimensional points in the data cube space are defined by dimension-value pairs. For example, the dimension-value pairs in ⟨time="Q1", location="Vancouver", item="computer"⟩ define a point in data cube space. A data cube **measure** is a numerical function that can be evaluated at each point in the data cube space. A measure value is computed for a given point by aggregating the data corresponding to the respective dimension-value pairs defining the given point. We will look at concrete examples of this shortly.

Measures can be organized into three categories, based on the kind of aggregate functions used.

- **distributive**: An aggregate function is *distributive* if it can be computed in a distributed manner as follows: Suppose the data is partitioned into n sets. The computation of the function on each partition derives one aggregate value. If the result derived by applying the function to the n aggregate values is the same as that derived by applying the function on all the data without partitioning, the function can be computed in a distributed manner. For example, `count()` can be computed for a data cube by first partitioning the cube into a set of subcubes, computing `count()` for each subcube, and then summing up the counts obtained for each subcube. Hence `count()` is a distributive aggregate function. For the same reason, `sum(), min()`, and `max()` are distributive aggregate functions. A measure is *distributive* if it is obtained by applying a distributive aggregate function.

- **algebraic**: An aggregate function is *algebraic* if it can be computed by an algebraic function with M arguments (where M is a bounded integer), each of which is obtained by applying a distributive aggregate function. For example, `avg()` (average) can be computed by `sum()/count()` where both `sum()` and `count()` are distributive aggregate functions. Similarly, it can be shown that `min_N(), max_N()`, and `standard_deviation()` are algebraic aggregate functions. A measure is *algebraic* if it is obtained by applying an algebraic aggregate function.

- **holistic**: An aggregate function is *holistic* if there is no constant bound on the storage size needed to describe a subaggregate. That is, there does not exist an algebraic function with M arguments (where M is a constant) that characterizes the computation. Common examples of holistic functions include `median()`, `mode()` (i.e., the most frequently occurring item(s)), and `rank()`. A measure is *holistic* if it is obtained by applying a holistic aggregate function.

Most large data cube applications require efficient computation of distributive and algebraic measures. Many efficient techniques for this exist. In contrast, it can be difficult to compute holistic measures efficiently. Efficient techniques to *approximate* the computation of some holistic measures, however, do exist. For example, instead of computing the exact `median()`, there are techniques which can estimate the approximate median value for a large data set with satisfactory results. In many cases, such techniques are sufficient to overcome the difficulties of efficient computation of holistic measures.

Example 2.7 Many measures of a data cube can be computed by relational aggregation operations. In Figure 2.4, we saw a star schema for *AllElectronics* sales which contains two measures, namely *dollars_sold* and *units_sold*. In Example 2.4, the *sales_star* data cube corresponding to the schema was defined using DMQL commands. *"But, how are these commands interpreted in order to generate the specified data cube?"*

Suppose that the relational database schema of *AllElectronics* is the following:

time(time_key, day, day_of_week, month, quarter, year)
item(item_key, item_name, brand, type)
branch(branch_key, branch_name, branch_type)
location(location_key, street, city, province_or_state, country)
sales(time_key, item_key, branch_key, location_key, number_of_units_sold, price)

The DMQL specification of Example 2.4 is translated into the following SQL query, which generates the required *sales_star* cube. Here, the `sum` aggregate function is used to compute both *dollars_sold* and *units_sold*.

```
select s.time_key, s.item_key, s.branch_key, s.location_key,
                    sum(s.number_of_units_sold * s.price), sum(s.number_of_units_sold)
from time t, item i, branch b, location l, sales s,
where s.time_key = t.time_key and s.item_key = i.item_key
        and s.branch_key = b.branch_key and s.location_key = l.location_key
group by s.time_key, s.item_key, s.branch_key, s.location_key
```

The cube created in the above query is the base cuboid of the *sales_star* data cube. It contains all of the dimensions specified in the data cube definition, where the granularity of each dimension is at the **join key** level. A join key is a key that links a fact table and a dimension table. The fact table associated with a base cuboid is sometimes referred to as the **base fact table**.

By changing the group by clauses, we may generate other cuboids for the *sales_star* data cube. For example, instead of grouping by *s.time_key*, we can group by *t.month*, which will sum up the measures of each group by month. Also, removing "group by *s.branch_key*" will generate a higher level cuboid (where sales are summed for all branches, rather than broken down per branch). Suppose we modify the above SQL query by removing *all* of the group by clauses. This will result in obtaining the total sum of *dollars_sold* and the total count of *units_sold* for the given data. This zero-dimensional cuboid is the apex cuboid of the *sales_star* data cube. In addition, other cuboids can be generated by applying selection and/or projection operations on the base cuboid, resulting in a lattice of cuboids as described in Section 2.2.1. Each cuboid corresponds to a different degree of summarization of the given data. □

Most of the current data cube technology confines the measures of multidimensional databases to *numerical data*. However, measures can also be applied to other kinds of data, such as spatial, multimedia, or text data. Techniques for this are discussed in Chapter 9.

2.2.5 Introducing concept hierarchies

"What is a concept hierarchy?"

A **concept hierarchy** defines a sequence of mappings from a set of low level concepts to higher level, more general concepts. Consider a concept hierarchy for the dimension *location*. City values for *location* include Vancouver, Montreal, New York, and Chicago. Each city, however, can be mapped to the province or state to which it belongs. For example, Vancouver can be mapped to British Columbia, and Chicago to Illinois. The provinces and states can in turn be mapped to the country to which they belong, such as Canada or the USA. These mappings form a concept hierarchy for the dimension *location*, mapping a set of low level concepts (i.e., cities) to higher level, more general concepts (i.e., countries). The concept hierarchy described above is illustrated in Figure 2.7.

Many concept hierarchies are implicit within the database schema. For example, suppose that the dimension *location* is described by the attributes *number, street, city, province_or_state, zipcode*, and *country*. These attributes are related by a total order, forming a concept hierarchy such as "*street* $<$ *city* $<$ *province_or_state* $<$ *country*". This hierarchy is shown in Figure 2.8a). Alternatively, the attributes of a dimension may be organized in a partial order, forming a **lattice**. An example of a partial order for the *time* dimension based on the attributes *day, week, month, quarter*, and *year* is "*day* $<$ $\{$*month* $<$*quarter; week*$\}$ $<$ *year*" [1]. This lattice structure is shown in Figure 2.8b). A concept hierarchy that is a total or partial order among attributes in a database schema is called a **schema hierarchy**. Concept hierarchies that are common to many applications may be predefined in the data mining system, such as the the concept hierarchy for *time*. Data mining systems should provide users with the flexibility to tailor predefined hierarchies according to their particular needs. For example, one may like to define a fiscal year starting on April 1, or an academic year starting on September 1.

Concept hierarchies may also be defined by discretizing or grouping values for a given dimension or attribute, resulting in a **set-grouping hierarchy**. A total or partial order can be defined among groups of values. An example of a set-grouping hierarchy is shown in Figure 2.9 for the dimension *price*.

There may be more than one concept hierarchy for a given attribute or dimension, based on different user viewpoints. For instance, a user may prefer to organize *price* by defining ranges for *inexpensive, moderately_priced*, and *expensive*.

[1] Since a *week* usually crosses the boundary of two consecutive months, it is usually not treated as a lower abstraction of *month*. Instead, it is often treated as a lower abstraction of *year*, since a year contains approximately 52 weeks.

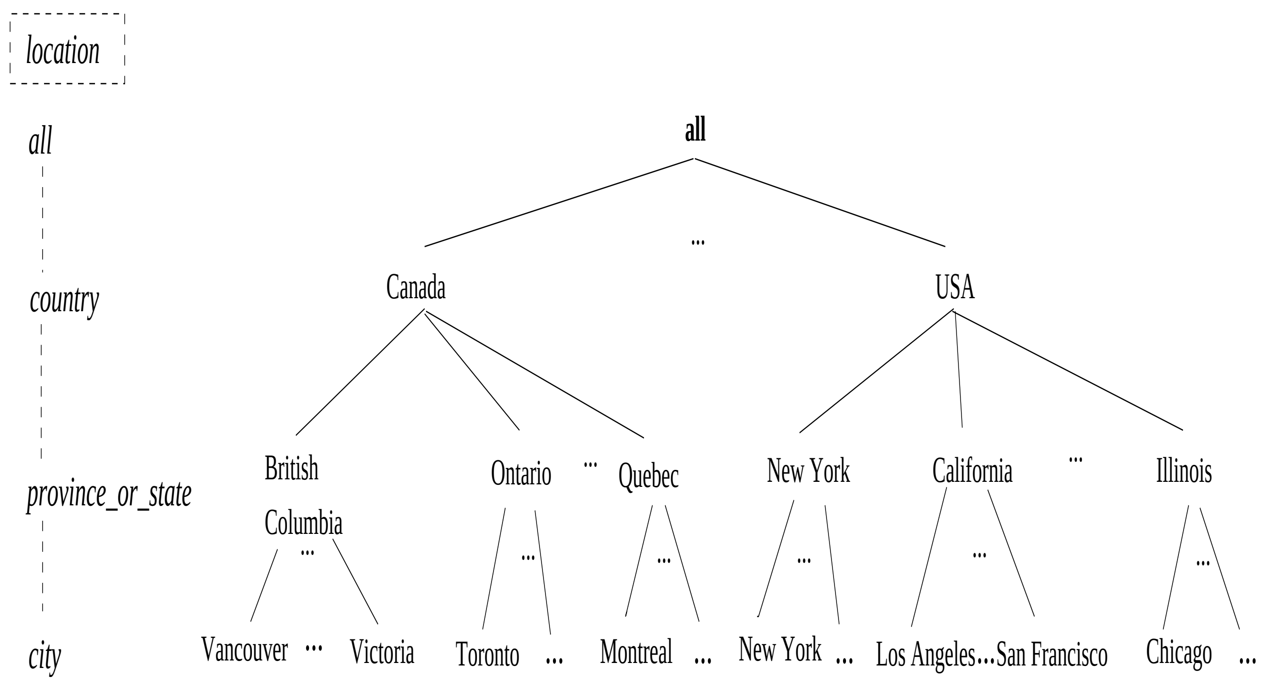


Figure 2.7: A concept hierarchy for the dimension *location*.

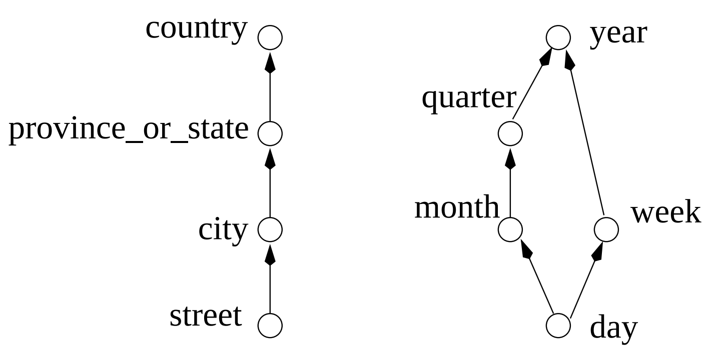
Concept hierarchies may be provided manually by system users, domain experts, knowledge engineers, or automatically generated based on statistical analysis of the data distribution. The automatic generation of concept hierarchies is discussed in Chapter 3. Concept hierarchies are further discussed in Chapter 4.

Concept hierarchies allow data to be handled at varying levels of abstraction, as we shall see in the following subsection.

2.2.6 OLAP operations in the multidimensional data model

"How are concept hierarchies useful in OLAP?"

In the multidimensional model, data are organized into multiple dimensions and each dimension contains multiple levels of abstraction defined by concept hierarchies. This organization provides users with the flexibility to view data from different perspectives. A number of OLAP data cube operations exist to materialize these different views, allowing interactive querying and analysis of the data at hand. Hence, OLAP provides a user-friendly environment



a) a hierarchy for *location* b) a lattice for *time*

Figure 2.8: Hierarchical and lattice structures of attributes in warehouse dimensions.

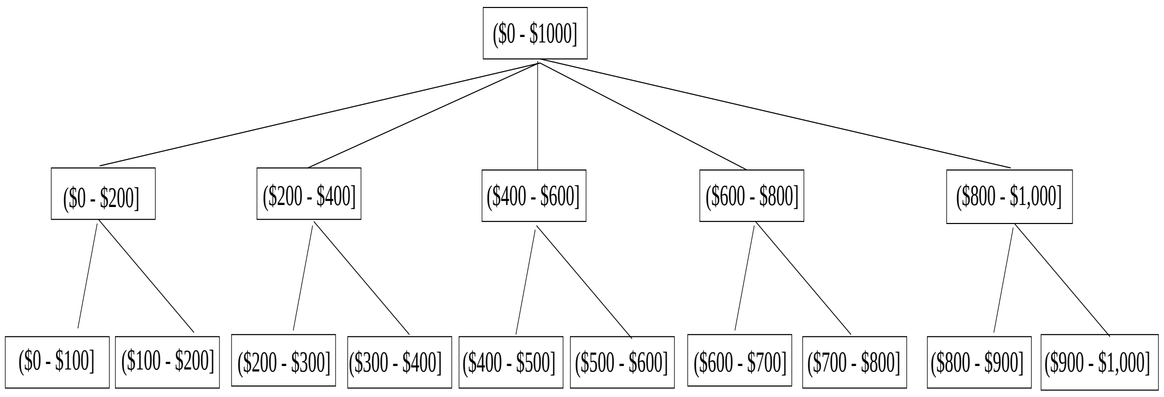


Figure 2.9: A concept hierarchy for the attribute *price*.

for interactive data analysis.

Example 2.8 Let's have a look at some typical OLAP operations for multidimensional data. Each of the operations described below is illustrated in Figure 2.10. At the center of the figure is a data cube for *AllElectronics* sales. The cube contains the dimensions *location, time*, and *item*, where *location* is aggregated with respect to city values, *time* is aggregated with respect to quarters, and *item* is aggregated with respect to item types. To aid in our explanation, we refer to this cube as the central cube. The data examined are for the cities Vancouver, Montreal, New York, and Chicago.

1. **roll-up**: The roll-up operation (also called the "drill-up" operation by some vendors) performs aggregation on a data cube, either by *climbing-up a concept hierarchy* for a dimension or by *dimension reduction*. Figure 2.10 shows the result of a roll-up operation performed on the central cube by climbing up the concept hierarchy for *location* given in Figure 2.7. This hierarchy was defined as the total order *street* < *city* < *province_or_state* < *country*. The roll-up operation shown aggregates the data by ascending the *location* hierarchy from the level of *city* to the level of *country*. In other words, rather than grouping the data by city, the resulting cube groups the data by country.

   When roll-up is performed by dimension reduction, one or more dimensions are removed from the given cube. For example, consider a sales data cube containing only the two dimensions *location* and *time*. Roll-up may be performed by removing, say, the *time* dimension, resulting in an aggregation of the total sales by location, rather than by location and by time.

2. **drill-down**: Drill-down is the reverse of roll-up. It navigates from less detailed data to more detailed data. Drill-down can be realized by either *stepping-down a concept hierarchy* for a dimension or *introducing additional dimensions*. Figure 2.10 shows the result of a drill-down operation performed on the central cube by stepping down a concept hierarchy for *time* defined as *day* < *month* < *quarter* < *year*. Drill-down occurs by descending the *time* hierarchy from the level of *quarter* to the more detailed level of *month*. The resulting data cube details the total sales per month rather than summarized by quarter.

   Since a drill-down adds more detail to the given data, it can also be performed by adding new dimensions to a cube. For example, a drill-down on the central cube of Figure 2.10 can occur by introducing an additional dimension, such as *customer_type*.

3. **slice and dice**: The *slice* operation performs a selection on one dimension of the given cube, resulting in a subcube. Figure 2.10 shows a slice operation where the sales data are selected from the central cube for the dimension *time* using the criteria *time="Q2"*. The *dice* operation defines a subcube by performing a selection on two or more dimensions. Figure 2.10 shows a dice operation on the central cube based on the following selection criteria which involves three dimensions: (*location="Montreal"* or *"Vancouver"*) and (*time="Q1"* or *"Q2"*) and (*item="home entertainment"* or *"computer"*).

4. **pivot (rotate)**: *Pivot* (also called "*rotate*") is a visualization operation which rotates the data axes in view in order to provide an alternative presentation of the data. Figure 2.10 shows a pivot operation where the

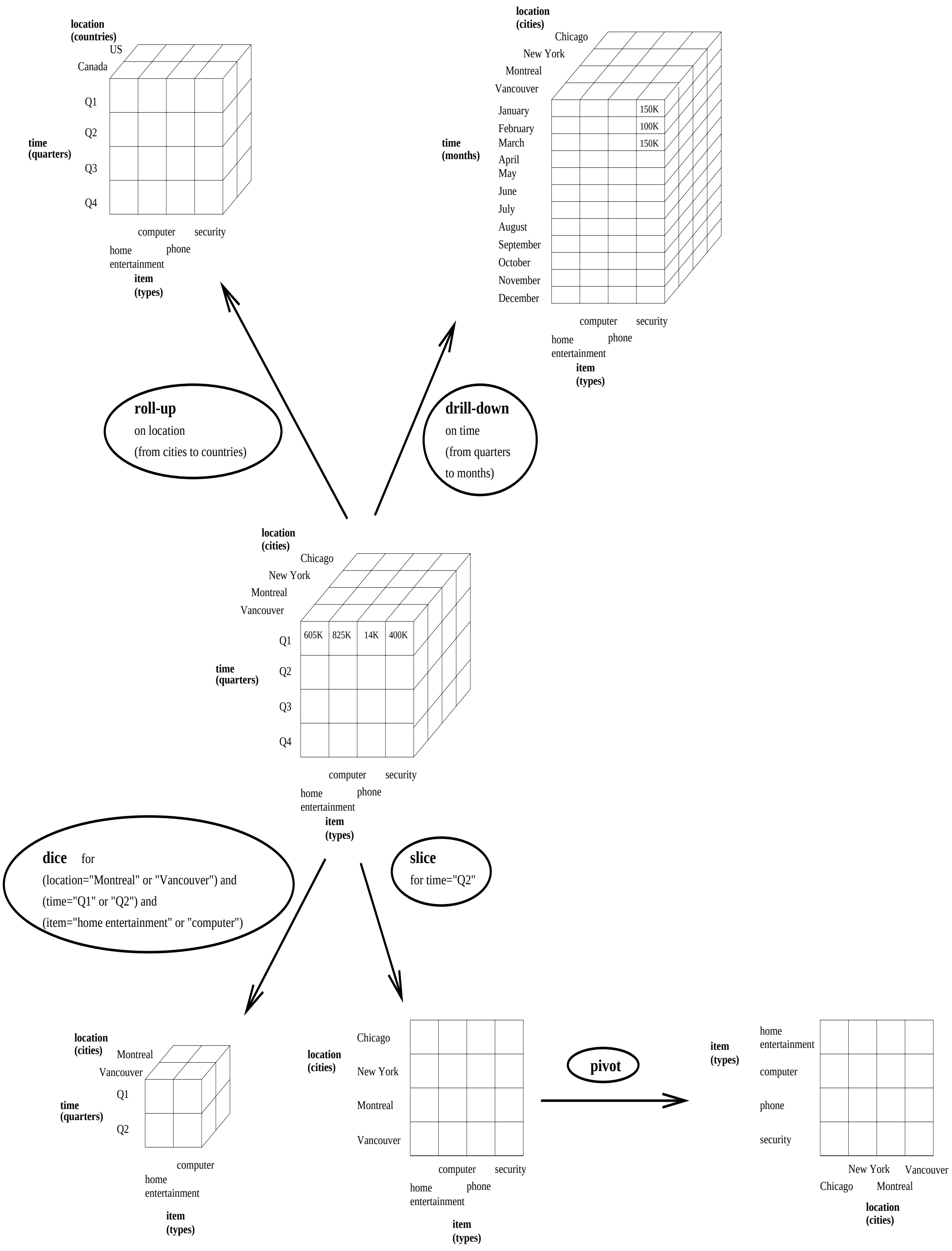


Figure 2.10: Examples of typical OLAP operations on multidimensional data.

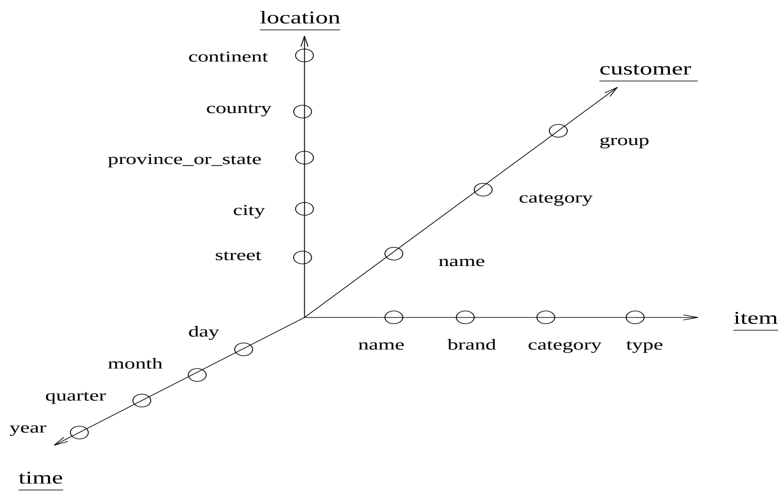


Figure 2.11: Modeling business queries: A starnet model.

item and *location* axes in a 2-D slice are rotated. Other examples include rotating the axes in a 3-D cube, or transforming a 3-D cube into a series of 2-D planes.

5. **other OLAP operations**: Some OLAP systems offer additional drilling operations. For example, **drill-across** executes queries involving (i.e., acrosss) more than one fact table. The **drill-through** operation makes use of relational SQL facilities to drill through the bottom level of a data cube down to its back-end relational tables.

   Other OLAP operations may include ranking the top-N or bottom-N items in lists, as well as computing moving averages, growth rates, interests, internal rates of return, depreciation, currency conversions, and statistical functions.

OLAP offers analytical modeling capabilities, including a calculation engine for deriving ratios, variance, etc., and for computing measures across multiple dimensions. It can generate summarizations, aggregations, and hierarchies at each granularity level and at every dimension intersection. OLAP also supports functional models for forecasting, trend analysis, and statistical analysis. In this context, an OLAP engine is a powerful data analysis tool.

2.2.7 A starnet query model for querying multidimensional databases

The querying of multidimensional databases can be based on a **starnet model**. A starnet model consists of radial lines emanating from a central point, where each line represents a concept hierarchy for a dimension. Each abstraction level in the hierarchy is called a **footprint**. These represent the granularities available for use by OLAP operations such as drill-down and roll-up.

Example 2.9 A starnet query model for the *AllElectronics* data warehouse is shown in Figure 2.11. This starnet consists of four radial lines, representing concept hierarchies for the dimensions *location, customer, item*, and *time*, respectively. Each line consists of footprints representing abstraction levels of the dimension. For example, the *time* line has four footprints: "day", "month", "quarter" and "year". A concept hierarchy may involve a single attribute (like *date* for the *time* hierarchy), or several attributes (e.g., the concept hierarchy for *location* involves the attributes *street, city, province_or_state*, and *country*). In order to examine the item sales at *AllElectronics*, one can roll up along the *time* dimension from *month* to *quarter*, or, say, drill down along the *location* dimension from *country* to *city*. Concept hierarchies can be used to **generalize** data by replacing low-level values (such as "day" for the *time* dimension) by higher-level abstractions (such as "year"), or to **specialize** data by replacing higher-level abstractions with lower-level values. □

2.3 Data warehouse architecture

2.3.1 Steps for the design and construction of data warehouses

The design of a data warehouse: A business analysis framework

"What does the data warehouse provide for business analysts?"

First, having a data warehouse may provide a *competitive advantage* by presenting relevant information from which to measure performance and make critical adjustments in order to help win over competitors. Second, a data warehouse can enhance business *productivity* since it is able to quickly and efficiently gather information which accurately describes the organization. Third, a data warehouse facilitates *customer relationship marketing* since it provides a consistent view of customers and items across all lines of business, all departments, and all markets. Finally, a data warehouse may bring about *cost reduction* by tracking trends, patterns, and exceptions over long periods of time in a consistent and reliable manner.

To design an effective data warehouse one needs to understand and analyze business needs, and construct a *business analysis framework*. The construction of a large and complex information system can be viewed as the construction of a large and complex building, for which the owner, architect, and builder have different views. These views are combined to form a complex framework which represents the top-down, business-driven, or owner's perspective, as well as the bottom-up, builder-driven, or implementor's view of the information system.

Four different views regarding the design of a data warehouse must be considered: the *top-down view*, the *data source view*, the *data warehouse view*, and the *business query view*.

- The **top-down view** allows the selection of the relevant information necessary for the data warehouse. This information matches the current and coming business needs.
- The **data source view** exposes the information being captured, stored, and managed by operational systems. This information may be documented at various levels of detail and accuracy, from individual data source tables to integrated data source tables. Data sources are often modeled by traditional data modeling techniques, such as the entity-relationship model or CASE (Computer Aided Software Engineering) tools.
- The **data warehouse view** includes fact tables and dimension tables. It represents the information that is stored inside the data warehouse, including precalculated totals and counts, as well as information regarding the source, date, and time of origin, added to provide historical context.
- Finally, the **business query view** is the perspective of data in the data warehouse from the view point of the end-user.

Building and using a data warehouse is a complex task since it requires *business skills, technology skills*, and *program management skills*. Regarding *business skills*, building a data warehouse involves understanding how such systems store and manage their data, how to build **extractors** which transfer data from the operational system to the data warehouse, and how to build **warehouse refresh software** that keeps the data warehouse reasonably up to date with the operational system's data. Using a data warehouse involves understanding the significance of the data it contains, as well as understanding and translating the business requirements into queries that can be satisfied by the data warehouse. Regarding *technology skills*, data analysts are required to understand how to make assessments from quantitative information and derive facts based on conclusions from historical information in the data warehouse. These skills include the ability to discover patterns and trends, to extrapolate trends based on history and look for anomalies or paradigm shifts, and to present coherent managerial recommendations based on such analysis. Finally, *program management skills* involve the need to interface with many technologies, vendors and end-users in order to deliver results in a timely and cost-effective manner.

The process of data warehouse design

"How can I design a data warehouse?"

A data warehouse can be built using a *top-down approach*, a *bottom-up approach*, or a *combination of both*. The **top-down approach** starts with the overall design and planning. It is useful in cases where the technology is mature and well-known, and where the business problems that must be solved are clear and well-understood. The

bottom-up approach starts with experiments and prototypes. This is useful in the early stage of business modeling and technology development. It allows an organization to move forward at considerably less expense and to evaluate the benefits of the technology before making significant commitments. In the **combined approach**, an organization can exploit the planned and strategic nature of the top-down approach while retaining the rapid implementation and opportunistic application of the bottom-up approach.

From the software engineering point of view, the design and construction of a data warehouse may consist of the following steps: *planning, requirements study, problem analysis, warehouse design, data integration and testing,* and finally *deployment of the data warehouse.* Large software systems can be developed using two methodologies: the *waterfall method* or the *spiral method.* The **waterfall method** performs a structured and systematic analysis at each step before proceeding to the next, which is like a waterfall, falling from one step to the next. The **spiral method** involves the rapid generation of increasingly functional systems, with short intervals between successive releases. This is considered a good choice for data warehouse development, especially for data marts, because the turn-around time is short, modifications can be done quickly, and new designs and technologies can be adapted in a timely manner.

In general, the warehouse design process consists of the following steps.

1. Choose a *business process* to model, e.g., orders, invoices, shipments, inventory, account administration, sales, and the general ledger. If the business process is organizational and involves multiple, complex object collections, a data warehouse model should be followed. However, if the process is departmental and focuses on the analysis of one kind of business process, a data mart model should be chosen.

2. Choose the *grain* of the business process. The grain is the fundamental, atomic level of data to be represented in the fact table for this process, e.g., individual transactions, individual daily snapshots, etc.

3. Choose the *dimensions* that will apply to each fact table record. Typical dimensions are time, item, customer, supplier, warehouse, transaction type, and status.

4. Choose the *measures* that will populate each fact table record. Typical measures are numeric additive quantities like *dollars_sold* and *units_sold.*

Since data warehouse construction is a difficult and long term task, its implementation scope should be clearly defined. The goals of an initial data warehouse implementation should be *specific, achievable*, and *measurable.* This involves determining the time and budget allocations, the subset of the organization which is to be modeled, the number of data sources selected, and the number and types of departments to be served.

Once a data warehouse is designed and constructed, the initial deployment of the warehouse includes initial installation, rollout planning, training and orientation. Platform upgrades and maintenance must also be considered. Data warehouse administration will include data refreshment, data source synchronization, planning for disaster recovery, managing access control and security, managing data growth, managing database performance, and data warehouse enhancement and extension. Scope management will include controlling the number and range of queries, dimensions, and reports; limiting the size of the data warehouse; or limiting the schedule, budget, or resources.

Various kinds of data warehouse design tools are available. **Data warehouse development tools** provide functions to define and edit metadata repository contents such as schemas, scripts or rules, answer queries, output reports, and ship metadata to and from relational database system catalogues. **Planning and analysis tools** study the impact of schema changes and of refresh performance when changing refresh rates or time windows.

2.3.2 A three-tier data warehouse architecture

"What is data warehouse architecture like?"

Data warehouses often adopt a three-tier architecture, as presented in Figure 2.12. The bottom tier is a **warehouse database server** which is almost always a relational database system. The middle tier is an **OLAP server** which is typically implemented using either (1) a **Relational OLAP (ROLAP)** model, i.e., an extended relational DBMS that maps operations on multidimensional data to standard relational operations; or (2) a **Multidimensional OLAP (MOLAP)** model, i.e., a special purpose server that directly implements multidimensional data and operations. The top tier is a **client**, which contains query and reporting tools, analysis tools, and/or data mining tools (e.g., trend analysis, prediction, and so on).

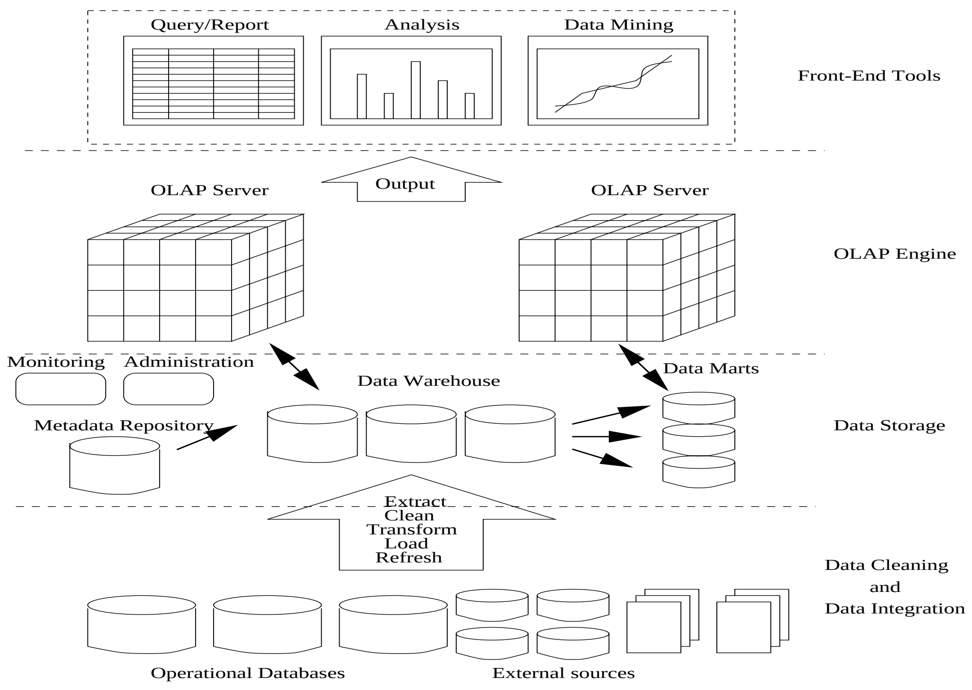


Figure 2.12: A three-tier data warehousing architecture.

From the architecture point of view, there are three data warehouse models: the *enterprise warehouse*, the *data mart*, and the *virtual warehouse*.

- **Enterprise warehouse**: An enterprise warehouse collects all of the information about subjects spanning the entire organization. It provides corporate-wide data integration, usually from one or more operational systems or external information providers, and is cross-functional in scope. It typically contains detailed data as well as summarized data, and can range in size from a few gigabytes to hundreds of gigabytes, terabytes, or beyond. An enterprise data warehouse may be implemented on traditional mainframes, UNIX superservers, or parallel architecture platforms. It requires extensive business modeling and may take years to design and build.

- **Data mart**: A data mart contains a subset of corporate-wide data that is of value to a specific group of users. The scope is confined to specific, selected subjects. For example, a marketing data mart may confine its subjects to customer, item, and sales. The data contained in data marts tend to be summarized.

  Data marts are usually implemented on low cost departmental servers that are UNIX-, Windows/NT-, or OS/2-based. The implementation cycle of a data mart is more likely to be measured in weeks rather than months or years. However, it may involve complex integration in the long run if its design and planning were not enterprise-wide.

  Depending on the source of data, data marts can be categorized into the following two classes:

  – *Independent* data marts are sourced from data captured from one or more operational systems or external information providers, or from data generated locally within a particular department or geographic area.

  – *Dependent* data marts are sourced directly from enterprise data warehouses.

- **Virtual warehouse**: A virtual warehouse is a set of views over operational databases. For efficient query processing, only some of the possible summary views may be materialized. A virtual warehouse is easy to build but requires excess capacity on operational database servers.

The top-down development of an enterprise warehouse serves as a systematic solution and minimizes integration problems. However, it is expensive, takes a long time to develop, and lacks flexibility due to the difficulty in achieving consistency and consensus for a common data model for the entire organization. The bottom-up approach to the design, development, and deployment of independent data marts provides flexibility, low cost, and rapid return

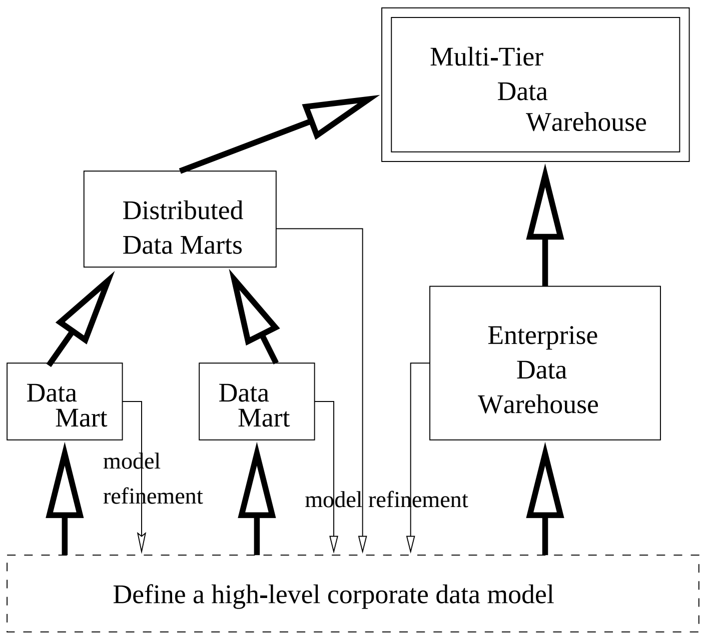

Multi-Tier Data Warehouse
Distributed Data Marts
Enterprise Data Warehouse
Data Mart
Data Mart
model refinement
model refinement
Define a high-level corporate data model

Figure 2.13: A recommended approach for data warehouse development.

of investment. It, however, can lead to problems when integrating various disparate data marts into a consistent enterprise data warehouse.

A recommended method for the development of data warehouse systems is to implement the warehouse in an incremental and evolutionary manner, as shown in Figure 2.13. First, a high-level corporate data model is defined within a reasonably short period of time (such as one or two months) that provides a corporate-wide, consistent, integrated view of data among different subjects and potential usages. This high-level model, although it will need to be refined in the further development of enterprise data warehouses and departmental data marts, will greatly reduce future integration problems. Second, independent data marts can be implemented in parallel with the enterprise warehouse based on the same corporate data model set as above. Third, distributed data marts can be constructed to integrate different data marts via hub servers. Finally, a **multi-tier data warehouse** is constructed where the enterprise warehouse is the sole custodian of all warehouse data, which is then distributed to the various dependent data marts.

2.3.3 OLAP server architectures: ROLAP vs. MOLAP vs. HOLAP

"What is OLAP server architecture like?"

Logically, OLAP engines present business users with multidimensional data from data warehouses or data marts, without concerns regarding how or where the data are stored. However, the physical architecture and implementation of OLAP engines must consider data storage issues. Implementations of a warehouse server engine for OLAP processing include:

- **Relational OLAP (ROLAP) servers**: These are the intermediate servers that stand in between a relational back-end server and client front-end tools. They use a relational or extended-relational DBMS to store and manage warehouse data, and OLAP middleware to support missing pieces. ROLAP servers include optimization for each DBMS back-end, implementation of aggregation navigation logic, and additional tools and services. ROLAP technology tends to have greater scalability than MOLAP technology. The DSS server of Microstrategy and Metacube of Informix, for example, adopt the ROLAP approach[2].

- **Multidimensional OLAP (MOLAP) servers**: These servers support multidimensional views of data through array-based multidimensional storage engines. They map multidimensional views directly to data

[2]Information on these products can be found at www.informix.com and www.microstrategy.com, respectively.

cube array structures. For example, Essbase of Arbor is a MOLAP server. The advantage of using a data cube is that it allows fast indexing to precomputed summarized data. Notice that with multidimensional data stores, the storage utilization may be low if the data set is sparse. In such cases, sparse matrix compression techniques (see Section 2.4) should be explored.

Many OLAP servers adopt a two-level storage representation to handle sparse and dense data sets: the dense subcubes are identified and stored as array structures, while the sparse subcubes employ compression technology for efficient storage utilization.

- **Hybrid OLAP (HOLAP) servers:** The hybrid OLAP approach combines ROLAP and MOLAP technology, benefitting from the greater scalability of ROLAP and the faster computation of MOLAP. For example, a HOLAP server may allow large volumes of detail data to be stored in a relational database, while aggregations are kept in a separate MOLAP store. The Microsoft SQL Server 7.0 OLAP Services supports a hybrid OLAP server.

- **Specialized SQL servers**: To meet the growing demand of OLAP processing in relational databases, some relational and data warehousing firms (e.g., Redbrick) implement specialized SQL servers which provide advanced query language and query processing support for SQL queries over star and snowflake schemas in a read-only environment.

The OLAP functional architecture consists of three components: the *data store*, the *OLAP server*, and the *user presentation module*. The data store can be further classified as a *relational data store* or a *multidimensional data store*, depending on whether a ROLAP or MOLAP architecture is adopted.

"So, how are data actually stored in ROLAP and MOLAP architectures?"

As its name implies, ROLAP uses relational tables to store data for on-line analytical processing. Recall that the fact table associated with a base cuboid is referred to as a *base fact table*. The base fact table stores data at the abstraction level indicated by the join keys in the schema for the given data cube. Aggregated data can also be stored in fact tables, referred to as **summary fact tables**. Some summary fact tables store both base fact table data and aggregated data, as in Example 2.10. Alternatively, separate summary fact tables can be used for each level of abstraction, to store only aggregated data.

Example 2.10 Table 2.4 shows a summary fact table which contains both base fact data and aggregated data. The schema of the table is "⟨*record_identifier (RID), item, location, day, month, quarter, year, dollars_sold* (i.e., sales amount)⟩", where *day, month, quarter*, and *year* define the date of sales. Consider the tuple with an RID of 1001. The data of this tuple are at the base fact level. Here, the date of sales is October 15, 1997. Consider the tuple with an RID of 5001. This tuple is at a more general level of abstraction than the tuple having an RID of 1001. Here, the "*Main Street*" value for *location* has been generalized to "*Vancouver*". The *day* value has been generalized to all, so that the corresponding *time* value is October 1997. That is, the *dollars_sold* amount shown is an aggregation representing the entire month of October 1997, rather than just October 15, 1997. The special value all is used to represent subtotals in summarized data.

RID	item	location	day	month	quarter	year	dollars_sold
1001	TV	Main Street	15	10	Q4	1997	250.60
...	...	...	...	...	...	...	...
5001	TV	Vancouver	all	10	Q4	1997	45,786.08
...	...	...	...	...	...	...	...

Table 2.4: Single table for base and summary facts.

□

MOLAP uses multidimensional array structures to store data for on-line analytical processing. For example, the data cube structure described and referred to throughout this chapter is such an array structure.

Most data warehouse systems adopt a client-server architecture. A relational data store always resides at the data warehouse/data mart server site. A multidimensional data store can reside at either the database server site or the client site. There are several alternative physical configuration options.

If a multidimensional data store resides at the client side, it results in a "fat client". In this case, the system response time may be quick since OLAP operations will not consume network traffic, and the network bottleneck happens only at the warehouse loading stage. However, loading a large data warehouse can be slow and the processing at the client side can be heavy, which may degrade the system performance. Moreover, data security could be a problem because data are distributed to multiple clients. A variation of this option is to partition the multidimensional data store and distribute selected subsets of the data to different clients.

Alternatively, a multidimensional data store can reside at the server site. One option is to set both the multidimensional data store and the OLAP server at the data mart site. This configuration is typical for data marts that are created by refining or re-engineering the data from an enterprise data warehouse.

A variation is to separate the multidimensional data store and OLAP server. That is, an OLAP server layer is added between the client and data mart. This configuration is used when the multidimensional data store is large, data sharing is needed, and the client is "thin" (i.e., does not require many resources).

2.3.4 SQL extensions to support OLAP operations

"How can SQL be extended to support OLAP operations?"

An OLAP server should support several data types including text, calendar, and numeric data, as well as data at different granularities (such as regarding the estimated and actual sales per item). An OLAP server should contain a calculation engine which includes domain-specific computations (such as for calendars) and a rich library of aggregate functions. Moreover, an OLAP server should include data load and refresh facilities so that *write* operations can update precomputed aggregates, and *write/load* operations are accompanied by data cleaning.

A multidimensional view of data is the foundation of OLAP. SQL extensions to support OLAP operations have been proposed and implemented in extended-relational servers. Some of these are enumerated as follows.

1. Extending the family of aggregate functions.

   Relational database systems have provided several useful aggregate functions, including `sum()`, `avg()`, `count()`, `min()`, and `max()` as SQL standards. OLAP query answering requires the extension of these standards to include other aggregate functions such as `rank()`, `N_tile()`, `median()`, and `mode()`. For example, a user may like to list the *top five most profitable items* (using `rank()`), list the *firms whose performance is in the bottom 10% in comparison to all other firms* (using `N_tile()`), or print *the most frequently sold items in March* (using `mode()`).

2. Adding reporting features.

   Many report writer softwares allow aggregate features to be evaluated on a time window. Examples include running totals, cumulative totals, moving averages, break points, etc. OLAP systems, to be truly useful for decision support, should introduce such facilities as well.

3. Implementing multiple group-by's.

   Given the multidimensional view point of data warehouses, it is important to introduce *group-by*'s for grouping sets of attributes. For example, one may want to list the total sales from 1996 to 1997 grouped by item, by region, and by quarter. Although this can be simulated by a set of SQL statements, it requires multiple scans of databases, and is thus a very inefficient solution. New operations, including cube and roll-up, have been introduced in some relational system products which explore efficient implementation methods.

2.4 Data warehouse implementation

Data warehouses contain huge volumes of data. OLAP engines demand that decision support queries be answered in the order of seconds. Therefore, it is crucial for data warehouse systems to support highly efficient cube computation techniques, access methods, and query processing techniques. *"How can this be done?"*, you may wonder. In this section, we examine methods for the efficient implementation of data warehouse systems.

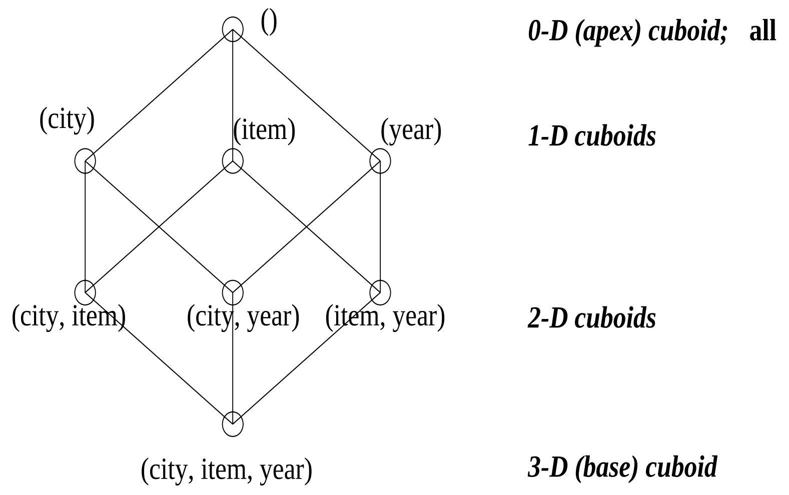


Figure 2.14: Lattice of cuboids, making up a 3-dimensional data cube. Each cuboid represents a different group-by. The base cuboid contains the three dimensions, *city, item*, and *year.*

2.4.1 Efficient computation of data cubes

At the core of multidimensional data analysis is the efficient computation of aggregations across many sets of dimensions. In SQL terms, these aggregations are referred to as **group-by's**.

The `compute cube` operator and its implementation

One approach to cube computation extends SQL so as to include a compute cube operator. The compute cube operator computes aggregates over all subsets of the dimensions specified in the operation.

Example 2.11 Suppose that you would like to create a data cube for *AllElectronics* sales which contains the following: *item, city, year*, and *sales_in_dollars.* You would like to be able to analyze the data, with queries such as the following:

1. "*Compute the sum of sales, grouping by item and city.*"
2. "*Compute the sum of sales, grouping by item.*"
3. "*Compute the sum of sales, grouping by city*".

What is the total number of cuboids, or group-by's, that can be computed for this data cube? Taking the three attributes, *city, item*, and *year*, as three dimensions and *sales_in_dollars* as the measure, the total number of cuboids, or group-by's, that can be computed for this data cube is $2^3 = 8$. The possible group-by's are the following: $\{(city, item, year)$, $(city, item)$, $(city, year)$, $(item, year)$, $(city)$, $(item)$, $(year)$, $()\}$, where () means that the group-by is empty (i.e., the dimensions are not grouped). These group-by's form a lattice of cuboids for the data cube, as shown in Figure 2.14. The base cuboid contains all three dimensions, *city, item*, and *year.* It can return the total sales for any combination of the three dimensions. The apex cuboid, or 0-D cuboid, refers to the case where the group-by is empty. It contains the total sum of all sales. Consequently, it is represented by the special value all.

□

An SQL query containing no group-by, such as "compute the sum of total sales" is a *zero-dimensional operation.* An SQL query containing one group-by, such as "compute the sum of sales, group by city" is a *one-dimensional operation.* A cube operator on n dimensions is equivalent to a collection of group by statements, one for each subset of the n dimensions. Therefore, the cube operator is the n-dimensional generalization of the group by operator.

Based on the syntax of DMQL introduced in Section 2.2.3, the data cube in Example 2.11, can be defined as

```
define cube sales [item, city, year]: sum(sales_in_dollars)
```