DATA WAREHOUSE ESSENTIALS

MICHAEL V. MANNINO

CHICAGO
BUSINESS PRESS

Detailed Table of Contents

Chapter 1

Data Warehouse Concepts and Design

Learning Objectives

This chapter explains the basic concepts and design practices of an emerging form of databases, called data warehouses, being used increasingly for business intelligence. After this chapter, the student should have acquired the following knowledge and skills:

- Explain conceptual differences between operational databases and data warehouses
- Understand architectures to apply data warehouse technology in organizations
- Understand the representation and manipulation of data cubes
- Apply relational data modeling patterns to multidimensional data
- Analyze data warehouse designs for summarizability problems and historical integrity
- Gain insights about the practices involved with enterprise data warehouse development

Overview

Imagine a corporate executive of a global electronics retailer asking the question, "What retail stores were the top producers during the past 12 months in major geographic regions?" Follow-up questions may include, "What were the most profitable products in the top producing stores?" and "What were the most successful product promotions at the top producing stores?" These questions are typical business intelligence questions, asked every day by managers all over the world. Answers to these questions often require complex SQL statements that may take hours to code and execute. Furthermore, formulating some of these queries may require data from a diverse set of internal legacy systems and external market sources, involving both relational databases and nonrelational data.

Business intelligence questions such as those in the previous paragraph pose new requirements for data modeling, database design methodologies, database infrastructure, and DBMS support. This chapter presents the management concepts and design practices of data warehouses to satisfy the requirements of business intelligence. In this chapter, you will initially learn about the unique requirements for data warehouse processing as opposed to the transaction processing requirements for operational databases. Then you will learn about the multidimensional data model, a conceptual approach for business analysts to use a data warehouse. You will then learn new data modeling and design skills that complement data modeling skills for operational databases. Finally, you will learn about methodologies to develop enterprise data warehouses spanning many functional areas of an organization.

This chapter provides the foundation to understand basic data warehouse concepts and development practices. Chapter 2 extends this foundation with query formulation skills and background on physical implementation of enterprise data warehouses. You will learn about extensions to relational DBMS products to support data warehouse processing and data integration practices to collect and transform data from many sources.

1.1 Basic Concepts

The type of data used for decision support purposes is conceptually different from that used in transaction processing databases. Consequently, databases that can store such data and computing architectures that can process such data are also different. This section examines these differences to provide a foundation for the detailed study of modeling and development practices in later sections.

1.1.1 Transaction Processing versus Business Intelligence

Transaction processing involves different needs in an organization than business intelligence. Transaction processing allows organizations to conduct daily business in an efficient manner. Operational or production databases used in transaction processing assist with decisions such as tracking orders, resolving customer complaints, and assessing staffing requirements. These decisions involve detailed data about business processes. In contrast, business intelligence processing helps management provide medium-term and long-term direction for an organization. Management needs support for decisions about capacity planning, product development, store location, product promotion, and other needs.

Historically, most organizations assumed that operational databases along with relational database technology could provide adequate support for business intelligence. As organizations developed operational databases for various functions, an information gap developed. Gradually, organizations and database vendors realized the limitations of operational databases and relational database technology for business intelligence.

Since the early 1990s, a consensus has emerged that operational databases must be transformed for business intelligence support. Operational databases can contain inconsistencies in areas such as formats, entity identification, frequency of update, and units of measure that hamper usage in business intelligence. In addition, business intelligence needs a broad view that integrates business processes. Because of the different requirements, operational databases are usually separate from databases for business intelligence. Using a common database for both kinds of processing can significantly degrade performance and make it difficult to summarize activity across business processes.

Commercial vendors have also performed substantial amounts of research and development to add features for business intelligence. Initially, a new breed of companies developed storage engines, summary data retrieval, and data transformation tools for business intelligence. Later relational database vendors added features for efficient management of summary data, query language extensions for summary data, optimization of queries involving summary data, and transformation of operational databases. The technology provided by both business intelligence firms and relational database vendors has rapidly matured to provide strong commercial solutions for developing and managing data warehouses.

1.1.2 Characteristics of Data Warehouses

Data warehouse, a term coined by William Inmon in 1990, refers to a central data repository where data from operational databases and other sources are integrated, cleaned, and standardized to support business intelligence. The transformational activities (cleaning, integrating, and standardizing) are essential for achieving benefits. Tangible benefits from a data warehouse can include increased revenue and reduced expenses enabled by business analysis that was not possible before the data warehouse was deployed. For example, a data warehouse may enable reduced losses due to improved fraud detection, improved customer retention through targeted marketing, and reduction in inventory carrying costs through improved demand forecasting.

Data Warehouse: a central repository for summarized and integrated data from operational databases and external data sources.

The processing requirements of decision support applications have led to four distinguishing characteristics for data warehouses, as described in the following list:

1. Subject-Oriented: A data warehouse is organized around major business subjects or entities such as customers, orders, and products. This subject orientation contrasts to the process orientation for transaction processing.

2. Integrated: Operational data from multiple databases and external data sources are integrated in a data warehouse to provide a single, unified database for business intelligence. Consolidation of data requires consistent naming conventions, uniform data formats, and comparable measurement scales across databases and external data sources.

3. Time-Variant: Data warehouses use time stamps to represent historical data. The time dimension is critical for identifying trends, predicting future operations, and setting operating targets. Data warehouses essentially consist of a long series of snapshots, each of which represents operational data captured at a point in time.

4. Nonvolatile: New data in a data warehouse are appended, rather than replaced, so that historical data are preserved. The act of appending new data is known as refreshing the data warehouse. Lack of update and delete operations ensures that update and deletion anomalies are not important for data warehouses. Transaction data are transferred to a data warehouse only when most updating activity has been completed.

Table 1-1 further depicts the characteristics of data warehouses compared to operational databases. Transaction processing relies on operational databases with current data at the individual level, while business intelligence processing utilizes data warehouses with historical data at both the individual and summarized levels. Individual-level data provides flexibility for responding to a wide range of business intelligence needs while summarized data provides fast response to repetitive queries. For example, an order-entry transaction requires data about individual customers, orders, and inventory items, while a business intelligence application may use monthly sales to customers over a period of several years. Operational databases therefore have a process orientation (e.g., all data relevant to a particular business process), compared to a subject orientation for data warehouses (e.g., all customer data or all order data). A transaction typically updates only a few rows, whereas a business intelligence application may query thousands to millions of rows.

Table 1-1: Comparison of Operational Databases and Data Warehouses

Characteristic	Operational Database	Data Warehouse
Currency	Current	Historical
Detail level	Individual	Individual and summary
Orientation	Process orientation	Subject orientation
Number of records processed	Few	Thousands
Normalization level	Mostly normalized	Frequent violations of BCNF
Update level	Volatile	Nonvolatile (refreshed)
Data model	Relational	Relational model with star schemas and multidimensional model with data cubes

Data integrity and usage patterns of transaction processing require that operational databases be highly normalized. In contrast, data warehouses are usually denormalized from Boyce-Codd Normal Form to reduce the effort to join large tables. Most data warehouse processing involves retrievals and periodic insertions of new data. These operations do not suffer from anomalies caused by a design that is not fully normalized.

Because of the different processing requirements, different data models have been developed for operational databases and data warehouses. The relational data model dominates for operational databases. In the early years of data warehouse deployment, the multidimensional data model dominated. In recent years, relational databases have been increasingly used for data warehouses with a schema pattern known as a star schema. The multidimensional data model is now typically used to provide a business analyst representation of a data warehouse.

1.1.3 Architectures for Data Warehouses

Despite the potential benefits of a data warehouse, many data warehouse projects have failed due to poor planning. Data warehouse projects are large efforts that involve coordination among many parts of an organization. Many organizations have underestimated the time and effort to reconcile different parts of a data warehouse. In addition, the sheer size of a data warehouse can lead to poor performance. An appropriate architecture can help alleviate problems with data warehouse performance and development efforts.

For small organizations, a two-tier data warehouse architecture is appropriate. In a two-tier data warehouse architec-

ture (Figure 1.1), operational data are transformed and then transferred to a data warehouse. A separate layer of servers may be used to support the complex activities of the transformation process. To assist with the transformation process, an enterprise data model (EDM) is created. The EDM describes the structure of the data warehouse and contains the metadata required to access operational databases and external data sources. The EDM may also contain details about cleaning and integrating data sources. Management uses the data warehouse directly to retrieve data for business intelligence.

Enterprise Data Model: a conceptual data model of the data warehouse defining the structure of the data warehouse and the metadata to access and transform operational databases and external data sources.

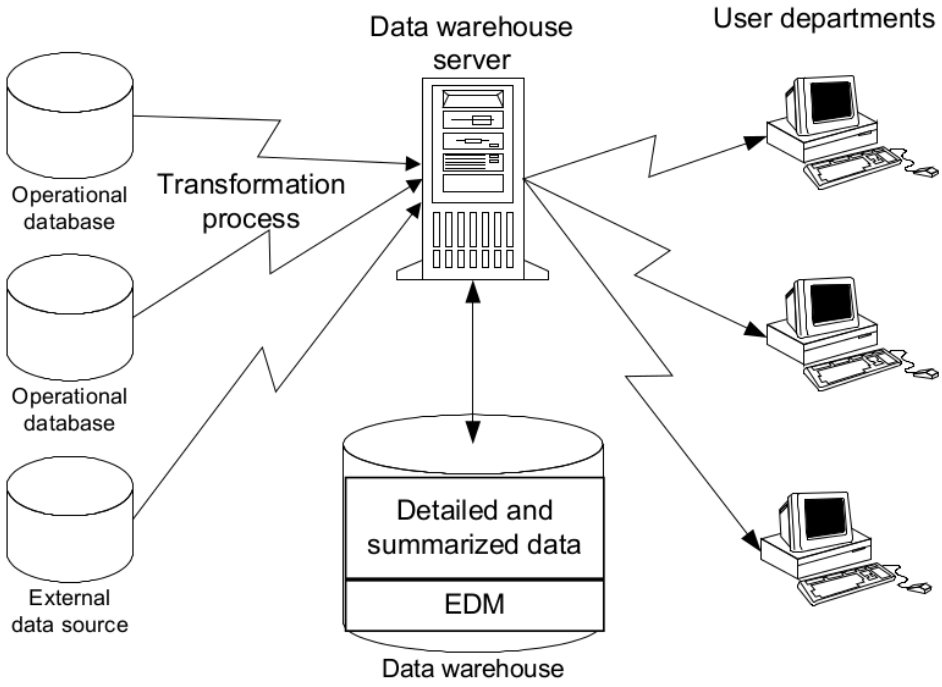


Figure 1.1: Two-Tier Data Warehouse Architecture

The two-tier architecture is a basic architecture for modest-sized data warehouses. For organizations with larger data warehouses and data-intensive applications for business intelligence, the three-tier data warehouse architecture as shown in Figure 1.2 provides a more scalable approach. Departmental users generally need access to small portions of the data warehouse, instead of the entire warehouse. To provide them with faster access while isolating them from data needed by other user groups, smaller data warehouses called data marts are often used. Data marts act as the interface between end users and the corporate data warehouse, storing a subset of the warehouse data and refreshing those data on a periodic (e.g., daily or weekly) basis. Generally, the data warehouse and the data marts reside on different servers to improve performance and fault tolerance. Departmental users retain control over their own data marts, while the data warehouse remains under the control of the corporate information systems staff.

Data Mart: a subset or view of a data warehouse, typically at a department or functional level, that contains all data required for business intelligence tasks of that department.

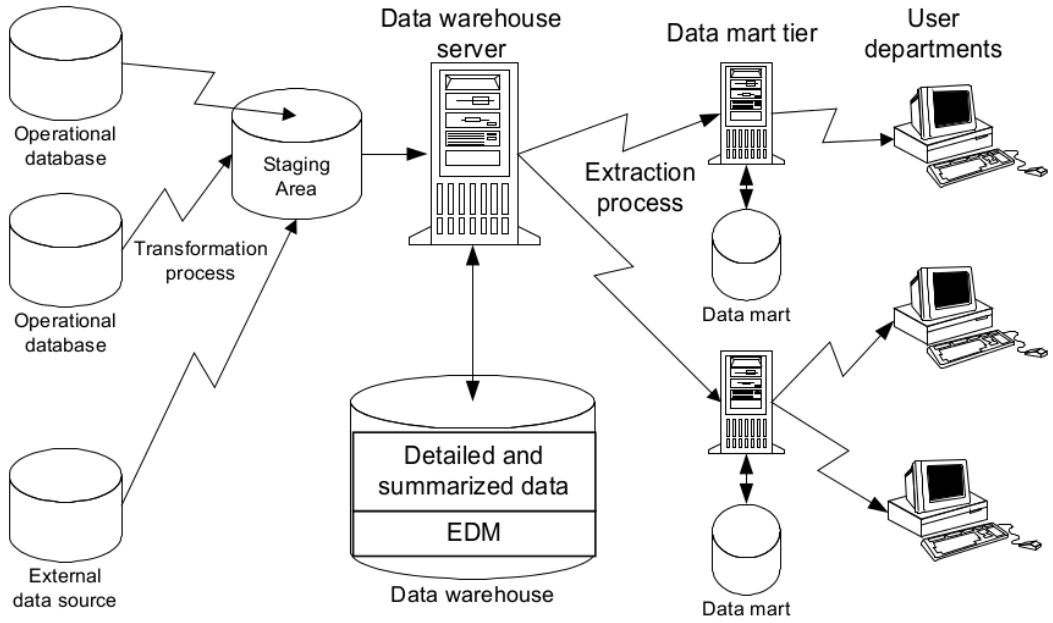


Figure 1.2: Three-Tier Data Warehouse Architecture with Staging Area

The three-tier architecture is sometimes augmented with a staging area to support the data transformation process. The staging area provides temporary storage of transformed data before loading into the data warehouse. The staging area is particularly useful for data warehouses with a large number of operational databases and external data sources requiring complex data transformations.

Given the enterprisewide context of a data warehouse, some organizations believe building a data warehouse may be unnecessarily delayed with lost business opportunities if too much emphasis is placed on first creating an enterprise data model. Instead, some organizations employ a bottom-up approach to data warehousing, as depicted in Figure 1.3. In a bottom-up data warehouse architecture, data are modeled one entity at a time and stored in separate data marts. Over time, new data are synthesized, cleaned, and merged into existing data marts or built into new data marts. The set of data marts may evolve into a large data warehouse if the organization can justify the expense of building an enterprise data model.

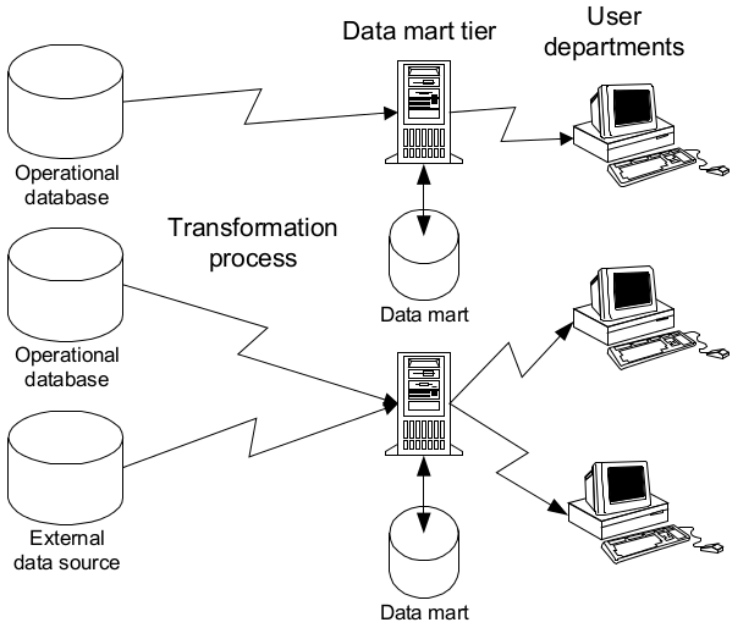


Figure 1.3: Bottom-Up Data Warehouse Architecture

A more recent development is the emergence of oper marts (short for operational mart) as noted by Imhoff (2001). An oper mart is a just-in-time data mart, usually built from one operational database in anticipation or in response to major events such as disasters and new product introductions. An oper mart supports peak demand for reporting and business analysis that accompanies a major event. After the decision support demand subsides, the oper mart may be dismantled or it may be merged into an existing data warehouse.

1.1.4 Data Mining

Data warehouses improve the quality of decision making by consolidating and aggregating transactional data. The value of a data warehouse can be increased if hidden patterns in the data can be discovered. Data mining refers to the process of discovering implicit patterns in data and using these patterns for business advantage. Data mining facilitates the ability to detect, understand, and predict patterns.

> **Data Mining**: the process of discovering implicit patterns in data stored in a data warehouse and using those patterns for business advantage.

The most common application of data mining techniques is target marketing. Mail-order companies can increase revenues and decrease costs if they can identify likely customers and eliminate customers not likely to purchase. Data mining techniques allow decision makers to focus marketing efforts by customer demographic and psychographic data. The retail, banking, travel, and consumer goods industries also have benefited from data mining techniques. For example, the retail industry uses data mining techniques to target promotions and change the mix and the arrangement of store items.

Data mining is best considered as an adjunct to a mature data warehouse. Data mining needs more detailed data than traditional data warehouses provide. The volumes of data and the dimensionality of data can be much greater for data mining techniques than other data warehouse analysis tools. Data mining techniques thrive with clean, high-dimensional, transaction data. To support these data mining requirements, many data warehouses now store detail data at the level of the individual customer, product, and so on.

Data mining requires a collection of tools that extend beyond traditional statistical analysis tools. Traditional statistical analysis tools are not well suited to high dimensional data with a mix of numeric and categorical data. In addition, traditional statistical techniques do not scale well to large amounts of data. Data mining typically includes the following kinds of tools:

- Data access tools to extract and sample transaction data according to complex criteria from large databases
- Data visualization tools that enable a decision maker to gain a deeper, intuitive understanding of data
- A rich collection of models to cluster, predict, and determine association rules from large amounts of data. The models involve neural networks, genetic algorithms, decision tree induction, rule discovery algorithms, probability networks, and other expert system technologies.
- An architecture that provides optimization, client-server processing, and parallel queries to scale to large amounts of data

As a complement to data warehousing, data mining provides insights that may elude traditional techniques. Data mining holds the promise of more effectively leveraging data warehouses by providing the ability to identify hidden relationships in the data. It facilitates data-driven discovery, using techniques such as building association rules (e.g., between advertising budget and seasonal sales), generating profiles (e.g., buying patterns for a specific customer segment), and so forth. This knowledge can be used to improve business operations in critical areas, enabling target marketing efforts, better customer service, and improved fraud detection.

1.1.5 Applications of Data Warehouses

Data warehousing projects have usually been undertaken for competitive reasons, to achieve strategic advantage or to stay competitive. In many industries, a few organizations have pioneered data warehousing technology to gain competitive advantage. Often a data warehousing project has been undertaken as part of a corporate strategy to shift from a product focus to a customer focus. Successful data warehouses have helped identify new markets, focus resources on profitable customers, improve retention of customers, and reduce inventory costs. After success by the pioneering organizations, other organizations have quickly followed to stay competitive.

Data warehousing projects have been undertaken in a wide range of industries. A few key applications have driven the adoption of data warehousing projects as listed in Table 1-2. Highly competitive industries such as retail, insurance, airlines, and telecommunications (particularly long-distance service) invested early in data warehouse technology and projects. Less competitive industries such as regulated utilities have been slower to invest although they are increasing investments as data warehouse technology and practice mature.

Table 1-2: Data Warehousing Applications by Industry

Industry	Key Applications
Airline	Yield management, route assessment
Telecommunications	Customer retention, network design
Insurance	Risk assessment, product design, fraud detection
Retail	Target marketing, supply-chain management

The maturity of data warehouse deployment varies among industries and organizations. Early adopters of data warehouses have deployed data warehouses since the early 1990s while later adopters have deployed data warehouses since the late 1990s. With the rapid development of data warehouse technology and best practices, continued investment in data warehouse technology and management practices are necessary to sustain business value. Many large organizations have made major redevelopments of their data warehouses to leverage improved technology and practices. To provide guidance about investment decisions and management practices, organizations are interested in comparisons to peer organizations to gauge the level of data warehouse usage.

The data warehouse maturity model has been proposed to provide guidance for data warehouse investment decisions (Eckerson 2007). The maturity model consists of six stages as summarized in Table 1-3. The stages provide a framework to view an organization's progress, not an absolute metric as organizations may demonstrate aspects of multiple stages at the same time. As organizations move from lower to more advanced stages, increased business value can occur. However, organizations may have difficulty justifying significant new data warehouse investments as benefits are sometimes difficult to quantify.

Table 1-3: Stages of the Data Warehouse Maturity Model

Stage	Scope	Architecture	Management Usage
Prenatal	System	Management reporting	Individual employees
Infant	Individual business analysts	Operational reports and spreadsheets (known as spreadmarts)	Management insight
Child	Departments	Data marts	Support business analysis
Teenager	Divisions	Data warehouses	Track business processes
Adult	Enterprise	Enterprise data warehouse	Drive organization
Sage	Inter-enterprise	Web services and external networks	Drive market and industry

Source: Eckerson 2007

An important insight of the maturity model is the difficulty of moving between certain stages. For small but growing organizations, moving from the infant to the child stages can be difficult because a significant investment in data warehouse technology is necessary. For large organizations, the struggle is to move from the teenager to the adult stage. To make the transition, upper management must perceive the data warehouse as a vital enterprise resource, not just a tool provided by the information technology department.

Data Warehouse Maturity Model: proposed to provide guidance for data warehouse investment decisions. The stages of the Data Warehouse Maturity Model provide a framework to view an organization's progress, not an absolute metric as organizations may demonstrate aspects of multiple stages at the same time.

The Data Warehouse Institute (TDWI) has developed a survey tool (available at tdwi.org) to assess the maturity of individual organizations in data warehouse development. The tool provides a maturity score in the categories of scope, sponsorship, funding, value, architecture, data, development, and delivery. The tool uses the scores to compare organizations based on industry, company size, budget, and other variables.

The Data Warehouse Maturity Model has been extended into the Business Intelligence (BI) Maturity Model by the Data Warehouse Institute. The BI Maturity Model adds dimensions of usage, insight, control, and business value to the adoption dimension of the original maturity model. As BI usage matures, organizations progress from information backlogs to utility support with access to power users, casual users, and external parties. Insight improves as BI adoption matures with organizations improving decision automation by increasing data freshness and decreasing decision latency. For control, organizations balance flexibility and standards, generally adopting more organization wide standards as maturity increases. Business value increases slowly as adoption matures with final stages of maturity providing competitive business advantage.

1.2 Multidimensional Representation of Data

After understanding the unique data requirements for decision support, you are ready to learn about technology to satisfy the requirements. The multidimensional data model supports data representation and operations specifically tailored for decision support processing in data warehouses. The multidimensional data model was originally proposed as a replacement for the relational model for data warehouses. Over time, the multidimensional model has evolved into a business analyst model, complementing the relational model for data warehouse storage. This section describes the terminology and operations of the multidimensional data model.

1.2.1 Example of a Multidimensional Data Cube

Consider a company that sells electronic products in different parts of the United States. In particular, the company markets four different printer products (mono laser, ink jet, photo, and portable) in five different states (California, Washington, Colorado, Utah, and Arizona). To store daily sales data for each product and each location in a relational database, you need Table 1-4 which consists of three columns (*Product*, *Location*, and *Sales*) and 20 rows (four instances of *Product* times five instances of *Location*).

The representation of Table 1-4 can be complex and unwieldy. First, imagine that the company wishes to add a fifth product (say, color laser). To track sales by states for this new product, you need to add five rows, one each for each state. Second, note that the data in Table 1-4 represents sales data for a particular day (for example, August 10, 2013). To store the same data for all 365 days of 2013, you need to add a fourth column to store the sales date, and duplicate the 20 rows for each date 365 times to yield a total of 7,300 rows. By the same token, if you wish to store historic data for a period of 10 years, you need 73,000 rows. Each new row must contain the product, state, and date values.

Table 1-4: Relational Representation of Sales Data

Product	Location	Sales
Mono Laser	California	80
Mono Laser	Utah	40
Mono Laser	Arizona	70
Mono Laser	Washington	75
Mono Laser	Colorado	65
Ink Jet	California	110
Ink Jet	Utah	90
Ink Jet	Arizona	55
Ink Jet	Washington	85
Ink Jet	Colorado	45
Photo	California	60
Photo	Utah	50
Photo	Arizona	60
Photo	Washington	45
Photo	Colorado	85
Portable	California	25
Portable	Utah	30
Portable	Arizona	35
Portable	Washington	45
Portable	Colorado	60

An examination of the data in Table 1-4 reveals that the data contain two dimensions, *Product* and *Location*, and a numeric value for the unit sales. Table 1-4 therefore can be conceptually simplified by rearranging the data in a multidimensional format (Table 1-5).

Table 1-5: Multidimensional Representation of Sales Data

Location	Product			
	Mono Laser	*Ink Jet*	*Photo*	*Portable*
California	80	110	60	25
Utah	40	90	50	30
Arizona	70	55	60	35
Washington	75	85	45	45
Colorado	65	45	85	60

The multidimensional representation is simple to understand and extend. For example, adding a fifth product category requires an additional column to the right of Table 1-4. Adding dates requires a third dimension called *Time*, resulting in a three-dimensional arrangement as shown in Figure 1.4. You can conceptually think of this three-dimensional table as a book consisting of 365 pages, each page storing sales data by product and state for a specific date of the

year. In addition, the multidimensional table is more compact because the row and column labels are not duplicated as in Table 1-3.

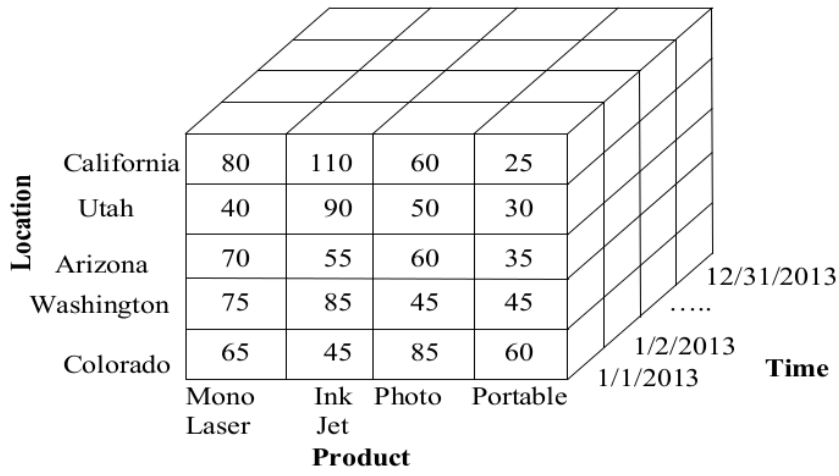


Figure 1.4: A Three-Dimensional Data Cube

The multidimensional representation also provides a convenient representation of summary totals. Each dimension in a cube can accommodate totals (row totals, column totals, depth totals, and overall totals) that a user can identify easily. For example, to add row totals to Table 1-5, a *Totals* column can be added with one value per row as shown in Table 1-6. In the relational representation as depicted in Table 1-4, totals must be added by using null values for column values. For example, to represent the total California sales for all products, the row <–, California, 275> should be added to Table 1-4 where – indicates all products.

Table 1-6: Multidimensional Representation of Sales Data with Row Totals

Location	Product				
	Mono Laser	*Ink Jet*	*Photo*	*Portable*	*Totals*
California	80	110	60	25	275
Utah	40	90	50	30	210
Arizona	70	55	60	35	220
Washington	75	85	45	45	250
Colorado	65	45	85	60	255

In addition to advantages in usability, the multidimensional representation can provide increased retrieval speed. Direct storage of multidimensional data obviates the need to convert from a table representation to a multidimensional representation. However, the multidimensional representation can suffer from excessive storage because many cells can remain empty. Even with compression techniques, large multidimensional tables can consume considerably more storage space than corresponding relational tables.

In summary, a multidimensional representation provides an intuitive interface for business analysts. As the number of dimensions increase, business analysts find a multidimensional representation easy to understand and visualize as compared to a relational representation. Because the multidimensional representation matches the needs of business analysts, this representation is widely used in business reporting tools even when relational tables provide physical storage.

1.2.2 Multidimensional Terminology

A data cube or hypercube generalizes the two-dimensional (Table 1-5) and three-dimensional (Figure 1.4) representations shown in the previous section. A data cube consists of cells containing measures (numeric values such as the unit

sales amounts) and dimensions to label or group numeric data (e.g., *Product*, *Location*, and *Time*). Each dimension contains values known as members. For instance, the *Location* dimension has five members (California, Washington, Utah, Arizona, and Colorado) in Table 1-6. Both dimensions and measures can be stored or derived. For example, purchase date is a stored dimension with purchase year, month, and day as derived dimensions.

Data Cube: a multidimensional format in which cells contain numeric data called measures organized by subjects called dimensions. A data cube is sometimes known as a hypercube because conceptually it can have an unlimited number of dimensions.

Dimension Details

Dimensions can have hierarchies composed of levels. For instance, the *Location* dimension may have a hierarchy composed of the levels country, state, and city. Likewise, the *Time* dimension can have a hierarchy composed of year, quarter, month, and date. Hierarchies can be used to drill down from higher levels of detail (e.g., country) to lower levels (e.g., state and city) and to roll-up in the reverse direction. Although hierarchies are not essential, they allow a convenient and efficient representation. Without hierarchies, the *Location* dimension must contain the most detailed level (city). However, this representation can be difficult to compute aggregates across the dimension. Alternatively, the *Location* dimension can be divided into separate dimensions for country, state, and city resulting in a larger data cube.

For flexibility, dimensions can have multiple hierarchies. In a dimension with multiple hierarchies, usually at least one level is shared. For example, the *Location* dimension can have one hierarchy with the levels country, state, and city, and a second hierarchy with the levels country, state, and postal code. The *Time* dimension can have one hierarchy with levels year, quarter, and date and a second hierarchy with levels year, week, and day of the year. Multiple hierarchies allow alternative organizations for a dimension.

Another dimension feature is the ragged hierarchy for a self-referencing relationship among members of the same level. For example, a manager dimension could have a ragged hierarchy to display relationships among managers and subordinates. An analyst manipulating a data cube may want to expand or contract the manager dimension according to the relationships among managers and subordinates.

The selection of dimensions has an influence on the sparsity of a data cube. Sparsity indicates the extent of empty cells in a data cube. Sparsity can be a problem if two or more dimensions are related. For example, if certain products are sold only in selected states, cells may be empty. If a large number of cells are empty, the data cube can waste space and be slow to process. Special compression techniques can be used to reduce the size of sparse data cubes.

Measure Details

Cells in a data cube contain measures such as the sales values in Figure 1.4. Measures support numeric operations such as simple arithmetic, statistical calculations, and optimization methods. A cell may contain one or more measures. For example, the number of units can be another measure for the sales data cube. The number of nonempty cells in a multidimensional cube should equal the number of rows in the corresponding relational table. For example, Table 1-5 contains 20 nonempty cells corresponding to 20 rows in Table 1-4.

Derived measures can be stored in a data cube or computed from other measures at run-time. Measures that can be derived from other measures in the same cell typically would not be stored. For example, total dollar sales can be calculated as total unit sales times the unit price measures in a cell. Summary measures derived from a collection of cells may be stored or computed depending on the number of cells and the cost of accessing the cells for the computation.

The aggregation property indicates allowable summary operations for measures. Additive measures can be summarized across all dimensions using addition. Common additive measures include sales, cost, and profit. For example, the sales measure in Figure 1.4 can be meaningfully summed across locations, products, and time periods. Semi-additive measures can be summarized in some dimensions but not all dimensions, typically not in the time dimension. Periodic measurements such as account balances and inventory levels are semi-additive. For example, account balance can be summed across product and location dimensions but not across time. However, account balances can be averaged across time because the average operation allocates account balances to dimension members. Non-additive measures

cannot be summarized in any dimension. Historical facts involving individual entities such as a unit price are non-additive. Some non-additive measures can be converted to additive or semi-additive. For example, extended price (unit price * quantity) is additive although unit price is not additive.

Aggregation Property: indicates allowable summary operations for measures. Business analysts who do not understand the allowable operations may perform operations that have no meaning. The aggregation property for a dimension can be additive (summarized on all dimensions using addition), semi-additive (summarized on some dimensions using addition), or non-additive (cannot be summarized on any dimension using addition).

Other Data Cube Examples

As this section has indicated, data cubes can extend beyond the three-dimensional example shown in Figure 1.4. Table 1-7 lists common data cubes to support human resource management and financial analysis. The dimensions with slashes indicate hierarchical dimensions. The time and location dimensions are also hierarchical, but possible levels are not listed since the levels can be organization specific.

Table 1-7: Data Cubes to Support Human Resource Management and Financial Analysis

Data Cube	Typical Dimensions	Typical Measures
Turnover analysis	Company/line of business/department, location, salary range, position classification, time	Head counts for hires, transfers, terminations, and retirements
Employee utilization	Company/line of business/department, location, salary range, position classification, time	Full time equivalent (FTE) hours, normal FTE hours, overtime FTE hours
Asset analysis	Asset type, years in service band, time, account, company/line of business/department, location	Cost, net book value, market value
Vendor analysis	Vendor, location, account, time, business unit	Total invoice amount

1.2.3 Time-Series Data

Time is one of the most common dimensions in a data warehouse and is useful for capturing trends, making forecasts, and so forth. A time series provides storage of all historic data in one cell, instead of specifying a separate time dimension. The structure of a measure becomes more complex with a time series, but the number of dimensions is reduced. In addition, many statistical functions can operate directly on time-series data.

A time series is an array data type with a number of special properties as listed below. The array supports a collection of values, one for each time period. Examples of time-series measures include weekly sales amounts, daily stock closing prices, and yearly employee salaries. The following list shows typical properties for a time series:

- Data Type: This property denotes the kind of data stored in the data points. The data type is usually numeric such as floating point numbers, fixed decimal numbers, or integers.
- Start Date: This property denotes the starting date of the first data point, for example, 1/1/2013.
- Calendar: This property contains the calendar year appropriate for the time series, for example, 2013 fiscal year. An extensive knowledge of calendar rules, such as determining leap years and holidays embedded in a calendar, reduces effort in data warehouse development.

- Periodicity: This property specifies the interval between data points. Periodicity can be daily, weekly, monthly, quarterly, yearly (calendar or fiscal years), hourly, 15-minute intervals, 4-4-5 accounting periods, custom periodicity, and so on.
- Conversion: This property specifies conversion of unit data into aggregate data. For instance, aggregating daily sales into weekly sales requires summation, while aggregating daily stock prices into weekly prices requires an averaging operation.

1.2.4 Data Cube Operators

A number of decision support operators have been proposed for data cubes. This section discusses the most commonly used operators. Most business analysis tools support additional operators along with convenient graphical interfaces for all operators.

Slice

Because a data cube can contain a large number of dimensions, users often need to focus on a subset of the dimensions to gain insights. The slice operator retrieves a subset of a data cube similar to the restrict operator of relational algebra. In a slice operation, one or more dimensions are set to specific values and the remaining data cube is displayed. For example, Figure 1.5 shows the data cube resulting from the slice operation on the data cube in Figure 1.4 where Time = 1/1/2013 and the other two dimensions (*Location* and *Product*) are shown.

A variation of the slice operator allows a decision maker to summarize across members rather than to focus on just one member. The slice-summarize operator replaces one or more dimensions with summary calculations. The summary calculation often indicates the total value across members or the central tendency of the dimension such as the average or median value. For example, Figure 1.6 shows the result of a slice-summarize operation where the *Product* dimension is replaced by the sum of sales across all products. A new column called *Total Sales* can be added to store overall product sales for the entire year.

Location	**Product**			
	Mono Laser	*Ink Jet*	*Photo*	*Portable*
California	80	110	60	25
Utah	40	90	50	30
Arizona	70	55	60	35
Washington	75	85	45	45
Colorado	65	45	85	60

(Location × Product Slice for Time = 1/1/2013)

Figure 1.5: Example Slice Operation

Location	**Time**			
	1/1/2013	*1/2/2013*	...	*Total Sales*
California	400	670	…	16,250
Utah	340	190	…	11,107
Arizona	270	255	…	21,500
Washington	175	285	…	20,900
Colorado	165	245	…	21,336

Figure 1.6: Example Slice-Summarize Operation

Dice

Because individual dimensions can contain a large number of members, users need to focus on a subset of members to gain insights. The dice operator replaces a dimension with a subset of values of the dimension. For example, Figure 1.7 shows the result of a dice operation to display sales for the State of Utah for January 1, 2013. A dice operation typically follows a slice operation and returns a subset of the values displayed in the preceding slice. It helps focus attention on one or more rows or columns of numbers from a slice.

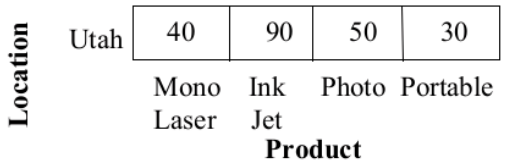


Figure 1.7: Example Dice Operation

Drill-Down

Users often want to navigate among the levels of hierarchical dimensions. The drill-down operator allows users to navigate from a more general level to a more specific level. For example, Figure 1.8 shows a drill-down operation on the State of Utah of the *Location* dimension. The plus sign by Utah indicates a drill-down operation.

Location	**Product**			
	Mono Laser	*Ink Jet*	*Photo*	*Portable*
California	80	110	60	25
+ Utah				
Salt Lake	20	20	10	15
Park City	5	30	10	5
Ogden	15	40	30	10
Arizona	70	55	60	35
Washington	75	85	45	45
Colorado	65	45	85	60

Figure 1.8: Drill-Down Operation for the State of Utah in Figure 1.5

Roll-Up

Roll-up (also called drill-up) is the opposite of drill-down. Roll-up involves moving from a specific level to a more general level of a hierarchical dimension. For example, a decision maker may roll-up sales data from daily to quarterly level for end-of-quarter reporting needs. In the printer sales example, Figure 1.5 shows a roll-up of the State of Utah from Figure 1.8.

Pivot

The pivot operator supports rearrangement of the dimensions in a data cube. For example, in Figure 1.8, the position of the *Product* and the *Location* dimensions can be reversed so that *Product* appears on the rows and *Location* on the columns. The pivot operator allows a data cube to be presented in the most appealing visual order.

The pivot operator is most typically used on data cubes of more than two dimensions. On data cubes of more than two dimensions, multiple dimensions appear in the row and/or column area because more than two dimensions cannot be displayed in other ways. For example, to display a data cube with *Location*, *Product*, and *Time* dimensions, the *Time* dimension can be displayed in the row area inside the *Location* dimension. A pivot operation could rearrange the data cube so that the *Location* dimension displays inside the *Time* dimension.

Summary of Operators

To help you recall the data cube operators, Table 1-8 summarizes the purpose of each operator.

Table 1-8: Summary of the Data Cube Operators

Operator	Purpose	Description
Slice	Focus attention on a subset of dimensions	Replace a dimension with a single member value or with a summary of its measure values
Dice	Focus attention on a subset of member values	Replace a dimension with a subset of members
Drill-down	Obtain more detail about a dimension	Navigate from a more general level to a more specific level of a hierarchical dimension
Roll-up	Summarize details about a dimension	Navigate from a more specific level to a more general level of a hierarchical dimension
Pivot	Allow a data cube to be presented in a visually appealing order	Rearrange the dimensions in a data cube

1.3 Relational Database Design for Data Warehouses

The multidimensional data model described in the previous section was originally implemented by special-purpose storage engines for data cubes. These multidimensional storage engines support the definition, manipulation, and optimization of large data cubes. Because of the commercial dominance of relational database technology, it was only a matter of time before relational DBMSs provided support for multidimensional data. In recent years, the major DBMS vendors have invested heavily in research and development to support multidimensional data. Because of the investment level and the market power of the relational DBMS vendors, most data warehouses now use relational DBMSs, at least in part.

Because of the importance of relational DBMS usage for data warehouses, this section presents relational database design principles for multidimensional data. The relational data modeling patterns based on the star schema are basic to relational database design for data warehouses. A relational database design should be checked for summarizability to ensure consistent and complete summary totals in user queries. Time representation is important for historical integrity in queries. Because most relational DBMSs lack dimension representation, the Oracle CREATE DIMENSION statement is demonstrated to depict support for dimensions beyond standard relational data modeling.

1.3.1 Relational Data Modeling Patterns

When using a relational database for a data warehouse, new data modeling patterns represent multidimensional data. A star schema is a data modeling representation of multidimensional data cubes. In a relational database, a star schema diagram looks like a star with one large central table, called the fact table, at the center of the star that is linked to multiple dimension tables in a radial manner. The fact table stores numeric data (facts), such as sales results, while the dimension tables store descriptive data corresponding to individual dimensions of the data cube such as product, location, and time. There is a 1-M relationship from each dimension table to the fact table.

Star Schema: a data modeling representation for multidimensional databases. In a relational database, a star schema has a fact table in the center related to multiple dimension tables in 1-M relationships.

Fact tables are classified based on the types of measures stored in the tables. A transaction table contains additive measures. Typical transaction tables store measures about sales, web activity, and purchases. A snapshot table provides a periodic view of an asset level. Typical snapshot tables store semi-additive measures about inventory levels, accounts receivable balances, and accounts payable balances. A factless table records event occurrences such as attendance, room reservations, and hiring. Typically, factless tables contain foreign keys without any measures. This classification is somewhat fluid as a fact table may be a combination of these types.

Figure 1.9 shows an ERD star schema for the sales example presented in Section 1.2. This ERD consists of four dimension entity types, *Item*, *Customer*, *Store*, and *TimeDim*, along with one (transaction) fact entity type called *Sales*. When converted to a table design, the *Sales* table has foreign keys to each dimension table (*Item*, *Customer*, *Store*, and *TimeDim*). The *Item* entity type provides data for the *Product* dimension shown in the Section 1.2 examples, while the

Store entity type provides data for the *Location* dimension. In some designs, the fact entity type depends on the related dimension entity types for its primary key. Since fact tables can have many relationships, it is generally preferred to have an artificial identifier rather than a large, combined primary key.

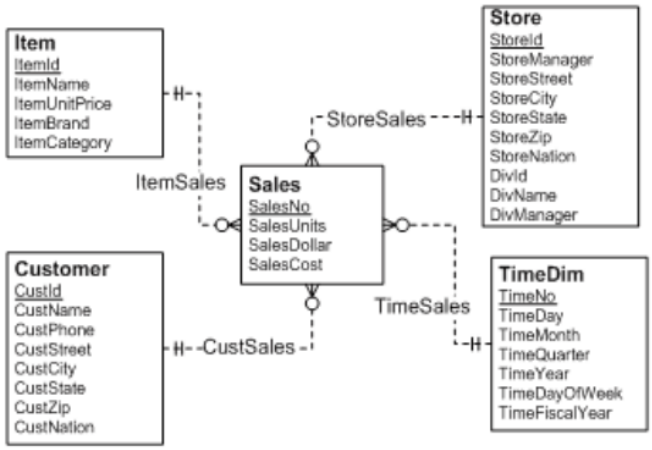


Figure 1.9: ERD Star Schema for the Store Sales Example

The store sales ERD in Figure 1.9 provides fine-grain detail for a data warehouse. The store sales ERD provides detail to the individual customer, store, and item. This level of detail is not necessary to support the sales data cubes presented in Section 1.2. However, a fine-grained level provides flexibility to support unanticipated analysis as well as data mining applications. This fine-grained level may replicate data in operational databases although the data warehouse representation may differ substantially because of the subject orientation of the data warehouse and the cleaning and integration performed on the source data.

Variations to the Star Schema

The star schema in Figure 1.9 represents only a single business process for sales tracking. Additional star schemas may be required for other processes such as shipping and purchasing. For related business processes that share some of the dimension tables, a star schema can be extended into a constellation schema with multiple fact entity types, as shown in Figure 1.10. When converted to a table design, the *Inventory* entity type becomes a fact table and 1-M relationships become foreign keys in the fact table. The *Inventory* entity type adds a number of measures including the quantity on hand of an item, the cost of an item, and the quantity returned. All dimension tables are shared among both fact tables except for the *Supplier* and *Customer* tables.

> **Constellation Schema**: a data modeling representation for multidimensional databases. In a relational database, a constellation schema contains multiple fact tables in the center related to dimension tables. Typically, the fact tables share some dimension tables.

Fact tables are usually normalized while dimension tables are often not in third normal form. For example, the *Store* entity type in Figures 1.9 and 1.10 is not in 3NF because *DivId* determines *DivName* and *DivManager*. Normalizing dimension tables to avoid storage anomalies is generally not necessary because they are usually stable and small. The nature of a data warehouse indicates that dimension tables should be designed for retrieval, not update. Retrieval performance is improved by eliminating the join operations that would be needed to combine fully normalized dimension tables.

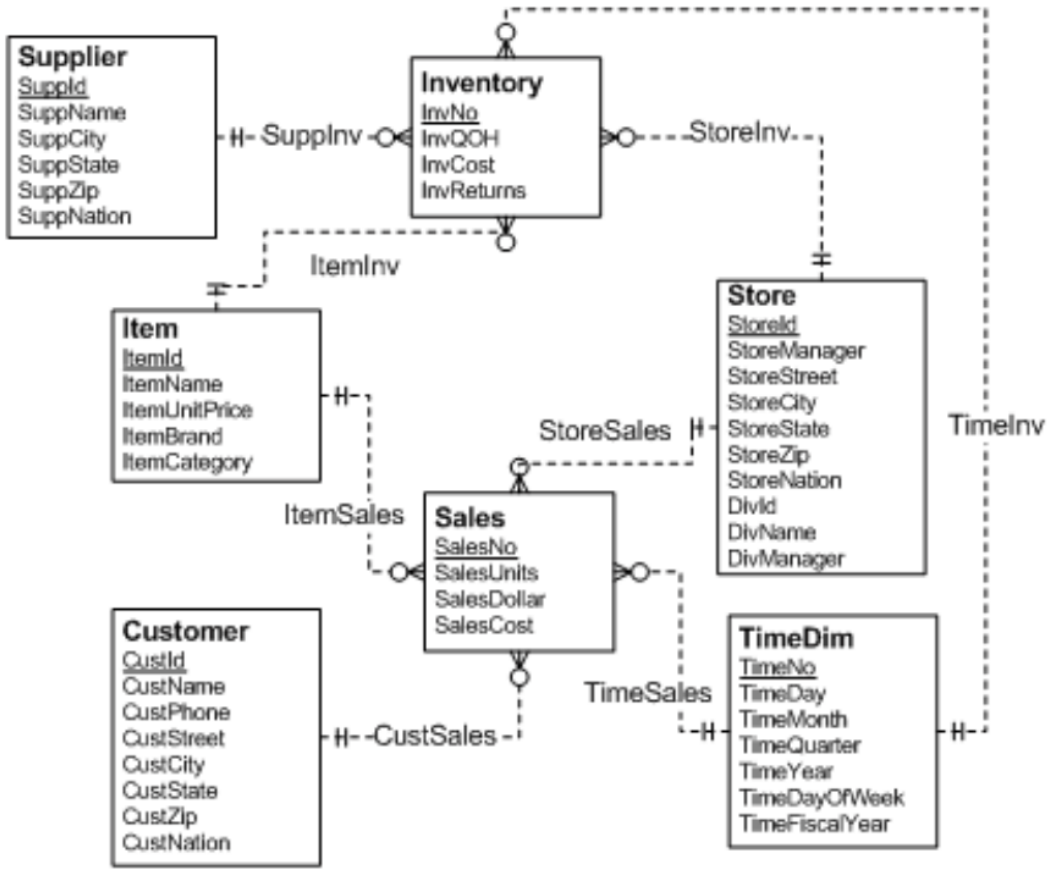


Figure 1.10: ERD Constellation Schema for the Sales-Inventory Example

When the dimension tables are small, denormalization provides only a small gain in retrieval performance. Thus, it is common to see small dimension tables normalized as shown in Figure 1.11. This variation is known as the snowflake schema because multiple levels of dimension tables surround the fact table. For the *Customer* and *Item* tables, full normalization may not be a good idea because these tables can contain a large number of rows.

Snowflake Schema: a data modeling representation for multidimensional databases. In a relational database, a snowflake schema has multiple levels of dimension tables related to one or more fact tables. You should consider the snowflake schema instead of the star schema for small dimension tables that are not in 3NF.

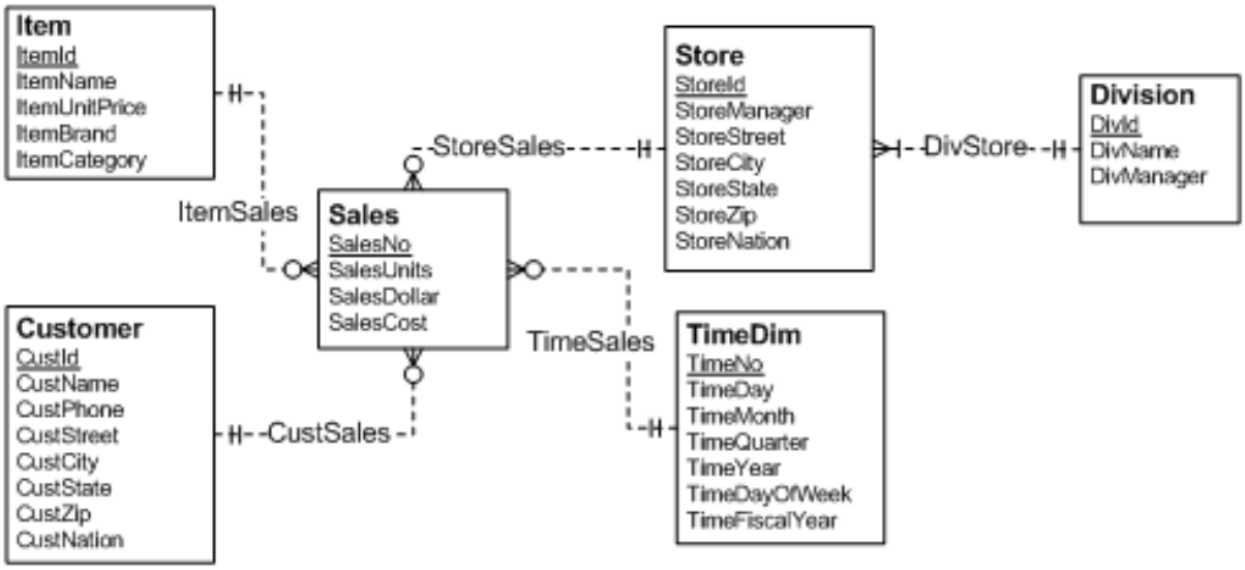


Figure 1.11: ERD Snowflake Schema for the Store Sales Example

The star schema and its variations require 1-M relationships from dimension tables to a fact table. The usage of 1-M relationships simplifies query formulation and supports optimization techniques discussed in Chapter 17. In some cases, M-N relationships may seem necessary between dimension and fact tables. Section 1.3.3 provides details about resolution of M-N relationships between fact and dimension tables.

1.3.2 Dimension Summarizability Problems and Patterns

Summarizability involves aggregation/disaggregation between a coarse level of detail and finer levels of details. Summary operations are common in data warehouse queries. Summary operations occur in drill down and rollup operations on a data cube and join operations combining fact and dimension tables.

Violations of summarizability conditions detract from the usability of a data warehouse. The most serious summarizability violations produce erroneous results. Even if results are not incorrect, violations of summarizability conditions can lead to user confusion. Violations of summarizability conditions (non summarizability) can also restrict the ability to use optimizations that improve query performance.

This section focuses on the summarizability problems involving dimension tables containing hierarchical dimensions. The next section covers problems involving join operations between fact and dimension tables.

Dimension Summarizability Problems

This subsection presents three dimension summarizability problems beginning with problems involving the drill-down operator. Drill-down incompleteness involves inconsistency between totals shown in drill-down operations. Drilling from a parent (coarser) level to a child (finer) level shows a smaller total indicating that measures attributed to parent members have not been allocated to child members. In Figure 1.12, the parent college level drills down to the department child level with a smaller total. The enrollment in the business college is omitted in the department level because the business college does not have departments. This inconsistency could cause user confusion and erroneous decision making.

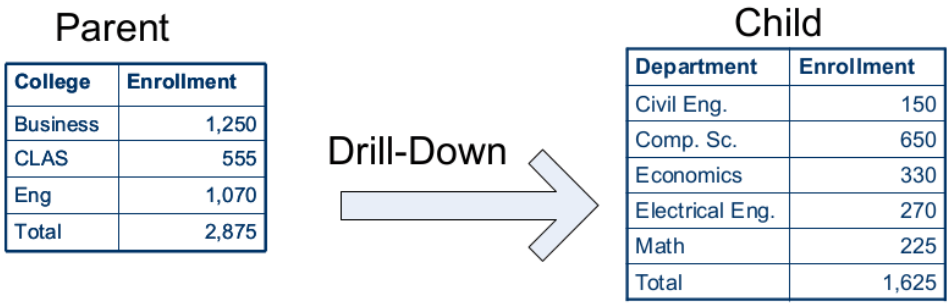


Figure 1.12: Drill-Down Incompleteness Example

Roll-up incompleteness is the reverse of drill-down incompleteness. Rolling up from a child (finer) level to a parent (coarser) level shows a smaller total indicating that measure values attributed to child members have not been allocated to parent members. In Figure 1.13, the product child level rolls up to the category parent level with a smaller total. The sales of napkin products are omitted in the category level because napkin products are not food or drink. This inconsistency could cause user confusion and erroneous decision making.

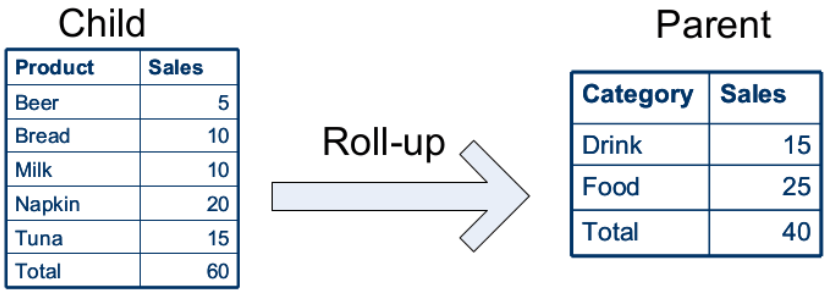


Figure 1.13: Roll-up Incompleteness Example

The non strict dimension problem involves M-N relationships between dimension levels, typically exceptions to 1-M relationships. For example, the time dimension can have M-N relationships between levels as depicted in Figure 1.14. The weeks of the year can overlap months leading to different totals for weeks and months. In Figure 1.14, January and February involve almost nine complete weeks. The week total (95) is more than the two month total (90). Users may perceive the difference as an inconsistency if they expect a 1-M relationship between month and weeks.

Dimension Summarizability Problems: inconsistent results that occur in summary operations involving relationships between entity types representing dimension levels. The inconsistent results involve different totals when summing values at the child and parent levels in a dimension hierarchy.

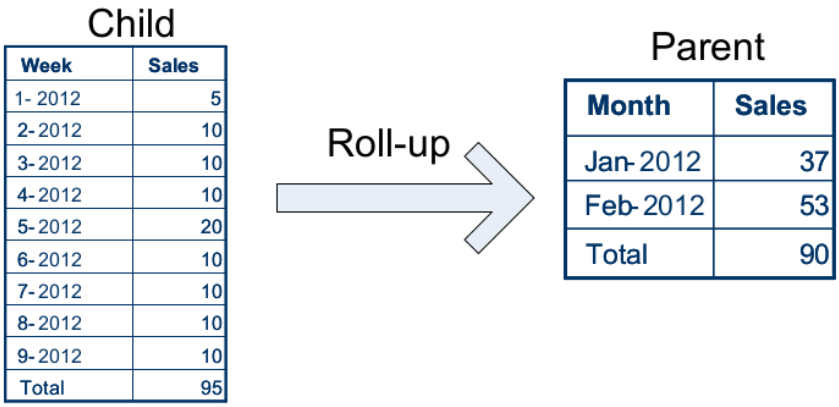


Figure 1.14: Example of a Non Strict Dimension Problem

Dimension Summarizability Patterns

To solidify your understanding of dimension summarizability, you should generalize beyond the examples presented in the previous subsection. Schema patterns provide a tool to generalize beyond examples. The summarizability patterns involve values for the minimum and maximum cardinalities. You should be able to recognize schema patterns that provide summarizability as well as patterns involving summarizability problems.

Summarizable schema patterns eliminate all three dimension summarizability problems. As shown in Figure 1.15, the regular dimension pattern involves a minimum cardinalilty of 1 for both the parent and child levels of a dimension hierarchy. The parent's minimum cardinality of 1 eliminates drill-down incompleteness, while the child's minimum cardinality of 1 eliminates rollup incompleteness. Although the unusual dimension pattern with a maximum cardinality of 1 for the parent eliminates summarizability problems, it is not typically seen in practice. The examples in Figure 1.16 involving Year-Month (a) and Division-Brand (b) are typical examples of regular summarizability patterns. In the year-month example, month refers to the month within a specific year such as August 2013. The relationship between *Division* and *Store* in Figure 1.11 is another example of a regular summarizability pattern.

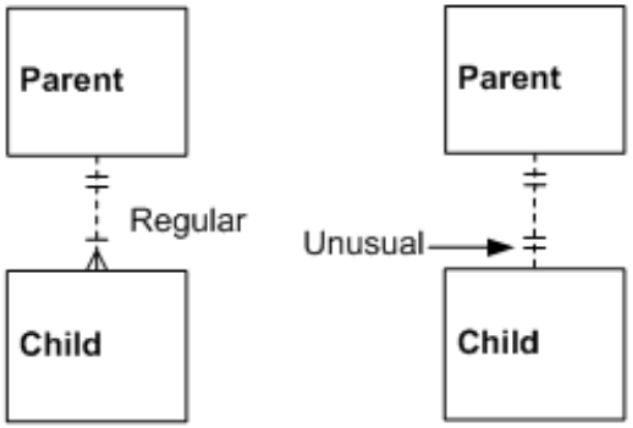


Figure 1.15: Schema Patterns for Summarizable Dimensions

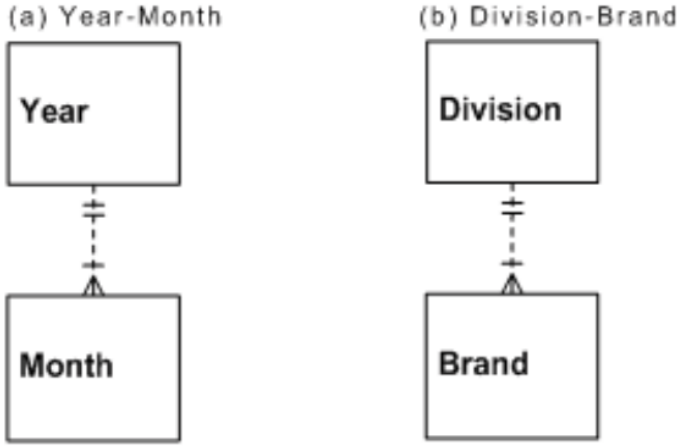


Figure 1.16: Examples of Schema Patterns for Summarizable Dimensions

Dimension Summarizability Pattern: a schema pattern that ensures consistent results in summary operations involving the parent and child entity type in the relationship. Relationship cardinalities determine the consistency of the summary operations. The regular and unusual patterns are the two dimension summarizability patterns

The values of minimum and maximum cardinalities determine schema patterns for the three dimension summarizability problems. The drill-down incomplete problem involves a minimum cardinality of 0 for the parent entity type as shown in Figure 1.17. The roll-up incomplete problem involves a child's minimum cardinality of 0. The non-strictness problem involves a M-N relationship. The examples in Figure 1.17 depict ERDs for the College-Department, Category-Product, and Month-WeekofYear examples previously presented in Figures 1.12 to 1.14 as small tables.

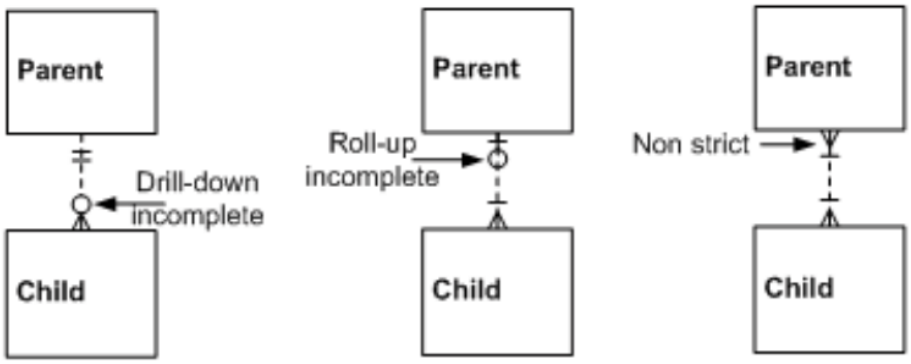


Figure 1.17: Schema Patterns for Dimension Summarizability Problems

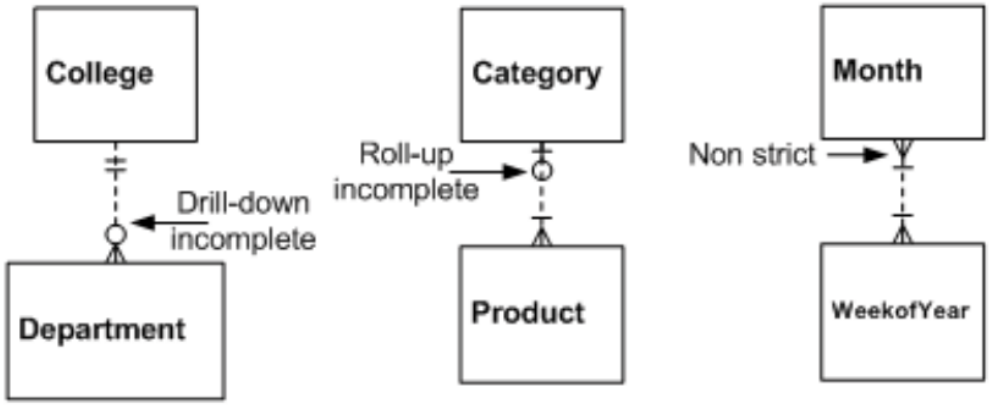


Figure 1.18: Examples of Schema Patterns for Dimension Summarizability Problems

Resolving the dimension summarizability problems is typically not difficult conceptually although the solutions may complicate the data integration process. For drill-down incompleteness, unallocated parent members should be related to a default child member. For example, colleges without departments should be represented by a new child member such as business college enrollments should be allocated to the "unallocated business" member in the department level. For roll-up incompleteness, a new default parent member should be used for child members without a parent. For example, a new "non food, non beverage" category can be added to the parent level to relate to unassociated child members such as napkin. For the non strict dimension problem, the simplest solution is to place the levels in different hierarchies. For example, the weekofyear and month levels should be placed in different time hierarchies. When the levels cannot be placed in different hierarchies, a major parent can be used in place of multiple, related parent members. For example, a product related to multiple categories can be related just to a major category. Another possible solution is to group related parent members into parent groups so that each child member is related to a single parent group.

To help you recall the conditions for dimension summarizability completeness and incompleteness, Table 1-9 lists the conditions for each pattern.

Table 1-9: Summary of the Dimension Summarizability Conditions

Summarizability Pattern	Conditions
Drill-down complete	Parent minimum cardinality = 1
Drill-down incomplete	Parent minimum cardinality = 0
Roll-up complete	Child minimum cardinality = 1
Roll-up incomplete	Child minimum cardinality = 0
Non strict	Child maximum cardinality = M
Regular	Parent min, max cardinality = (1, M)
	Child min, max cardinality = (1, 1)
Unusual	Parent max cardinality = 1

1.3.3 Dimension-Fact Summarizability Problems and Patterns

Relationships between dimension and fact tables dominate relational representation of data warehouses as explained in Section 1.3.1. Thus, it is important that join operations between fact and dimension tables show consistent summaries of measures. Incomplete dimension-fact relationships involve fact entities that do not have a related parent entity in a dimension-fact relationship. Inconsistencies caused by incomplete relationships are manifest in join operations with summary calculations. Figure 1.19 demonstrates inconsistent totals between (a) sales summarized by both customer and month and (b) sales summarized by month only. The sales summarized by both customer and month are inconsistent with the sales summarized by month due to anonymous customers. Some sales have been recorded without a known customer due to customer requirements for anonymity. A business analyst may become confused because of the inconsistent totals.

(a) Summarized by Customer and Month

Customer	Month	Sales
Cust-1	Jan-2013	10
Cust-2	Jan-2013	5
Cust-3	Feb-2013	15
Total		30

(b) Summarized by Month Only

Month	Sales
Jan-2013	25
Feb-2013	15
Total	40

Figure 1.19: Incomplete Dimension-Fact Relationship Example

The non strict dimension-fact relationship problem involves double counting measure values due to a M-N relationship between dimension and fact tables. In Figure 1.20, double counting of sales occurs when summarizing sales by individual salesperson and date. The sale on February 10, 2013 involves both SP1 and SP2. The individual sales total (55) is greater than the shared sales (45) because the shared sale is counted twice in Figure 1.20(a). The M-N relationship between the salesperson dimension table and sales fact table leads to double counting of the sales total, possibly causing erroneous decision making and user confusion.

(a) Unit sales by salesperson

Salesperson	Date	UnitSales
SP1	10-Feb-2013	10
SP2	10-Feb-2013	10
SP3	11-Feb-2013	15
SP4	12-Feb-2013	20
Total		55

(b) Shared unit sales by salesperson

Salesperson	Date	UnitSales
SP1, SP2	10-Feb-2013	10
SP3	11-Feb-2013	15
SP4	12-Feb-2013	20
Total		45

Figure 1.20: Non Strict Dimension-Fact Relationship Example

Dimension-Fact Relationship Summarizability Problems: inconsistent results that occur in summary operations involving relationships between dimension and fact entity types. The inconsistent results involve different totals when summing values involving different dimension-fact relationships.

Schema patterns for dimension-fact relationships support a more general understanding of summarizability problems. Figure 1.21 depicts patterns for summarizable dimension-fact relationships. In the regular dimension-fact pattern, a dimension entity may not be related to any fact entity. The optional relationship supports sparsity in which some dimension members do not have related fact entities. In Figure 1.21, some products have not been sold so the relationship is optional. The unusual dimension-fact pattern involves a mandatory relationship for the dimension entity type. In Figure 1.22, every date must have a related sale. The mandatory relationship for the dimension entity type is often not enforced because of difficulties in the data integration process.

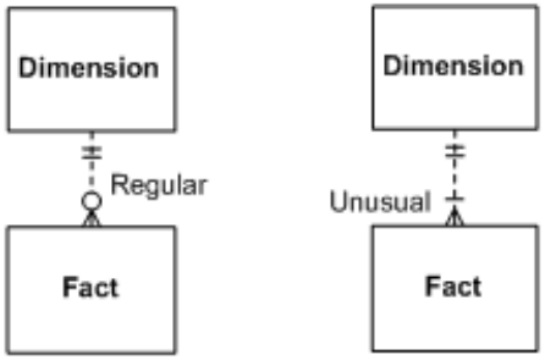


Figure 1.21: Schema Patterns for Summarizable Dimension-Fact Relationships

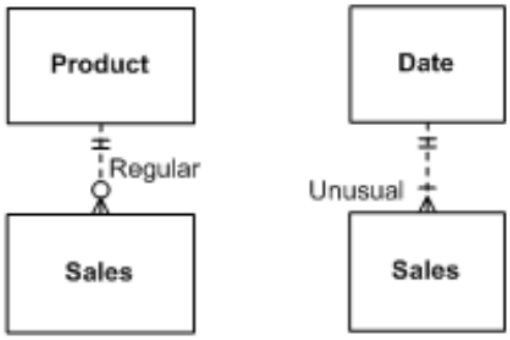


Figure 1.22: Schema Examples of Summarizable Dimension-Fact Relationships

Dimension-Fact Relationship Summarizability Pattern: a schema pattern that ensures consistent results in summary operations involving relationships between dimension and fact entity types. Relationship cardinalities determine the consistency of the summary operations. The regular and unusual patterns are the two dimension-fact relationship summarizability patterns.

Non summarizable dimension-fact relationship patterns deviate from the summarizable patterns in the cardinalities for the fact entity type. The incomplete dimension-fact relationship pattern involves a minimum cardinality of 0 for the fact entity type as shown in Figure 1.23. As an example, the *Purchase* entity type in Figure 1.24 has a minimum cardinality of 0 indicating an incomplete dimension-fact relationship. The non strict dimension-fact relationship involves a minimum cardinality of M for the fact entity type as shown in Figure 1.23. As an example, the *Sales* entity type has a maximum cardinality of M in Figure 1.24.

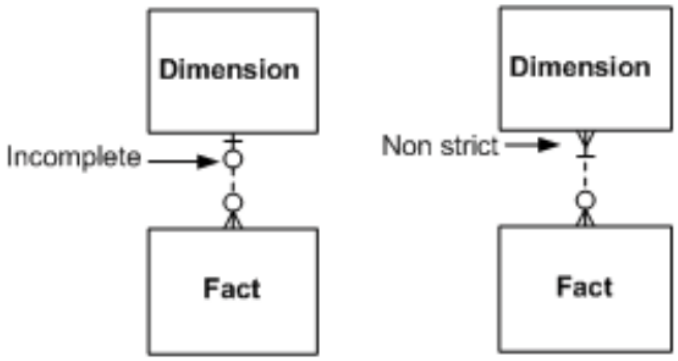


Figure 1.23: Schema Patterns for Non Summarizable Dimension-Fact Relationships

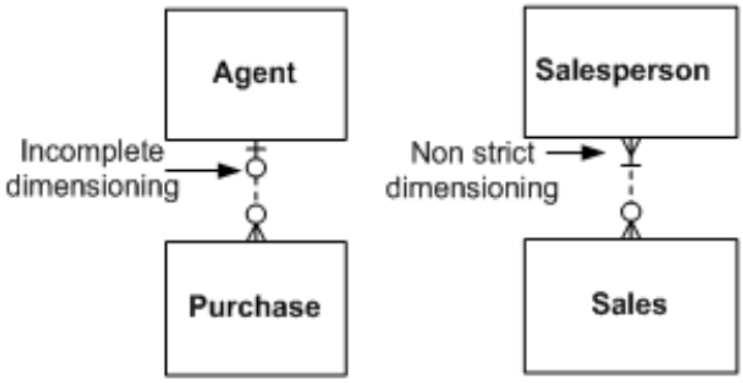


Figure 1.24: Schema Examples of Non Summarizable Dimension-Fact Relationships

Resolving incomplete dimension-fact relationships is conceptually simple although the resolution may complicate the data integration process. Unrelated fact entities should be connected to a default dimension entity if a connection to an actual dimension entity cannot be made. For example, anonymous sales should be connected to a default anonymous customer in the customer entity type.

Resolving non strict dimension-fact relationships can be more complex than resolution of incomplete relationships. Sometimes the source data has exceptions that involve M-N relationships, not 1-M relationships. For example, if the *Sales* fact table is derived from customer invoices, some invoices may involve multiple customers such as roommates or spouses. Two ways to resolve non strict dimension-fact relationships are explained in the following list.

- If there are a small, fixed number of possible related entities, a simple adjustment can be made to a fact entity type. Multiple relationships can be added between dimension and fact entity types to allow for more than one related entity. For example, the *Sales* entity type in Figure 1.11 can have an additional relationship to identify an optional second customer on an invoice.

- If there can be groups of related entities, the representation is more difficult. An entity type representing a group of related entities can be added with an associative entity type that connects the other entity types. For example to represent customer groups in Figure 1.11, a customer group entity type can be added along with 1-M relationships to connect the customer group, *Sales*, and *Customer* entity types.

To help you recall the conditions for summarizability completeness and strictness of dimension-fact relationships, Table 1-10 lists the conditions for each pattern.

Table 1-10: Summary of Dimension-Fact Relationship Summarizability Conditions

Summarizability Pattern	Conditions
Complete dimension-fact relationship	Fact minimum cardinality = 1
Incomplete dimension-fact relationship	Fact minimum cardinality = 0
Strict dimension-fact relationship	Fact maximum cardinality = 1
Non strict dimension-fact relationship	Fact maximum cardinality = M
Regular dimension-fact relationship	Dimension min, max cardinality = (0, M) Fact min, max cardinality = (1, 1)
Unusual dimension-fact relationship	Dimension minimum cardinality = 1

1.3.4 Time Representation in Star Schemas

Time representation is a crucial issue for data warehouses because most queries use time in conditions. The principal usage of time is to record the occurrence of facts. The simplest representation is a timestamp data type for a column in a fact table. In place of a timestamp column, many data warehouses use a foreign key to a time dimension table as shown in Figures 1.9 to 1.11. Using a time dimension table supports convenient representation of organization-specific calendar features such as holidays, fiscal years, and week numbers that are not represented in timestamp data types. The granularity of the time dimension table is usually in days. If time of day is also required for a fact table, it can be added as a column in the fact table to augment the foreign key to the time table.

Most fact tables involve time represented as a foreign key to the time table with augmentation for time of day if required. For fact tables involving international operations, two time representations (time table foreign keys along with optional time of day columns) can be used to record the time at source and destination locations. A variation identified by Kimball (2003) is the accumulating fact table that records the status of multiple events rather than one event. For example, a fact table containing a snapshot of order processing would include order date, shipment date, delivery date, payment date, and so on. Each event occurrence column can be represented by a foreign key to the time table along with a time of day column if needed.

For dimension tables, time representation involves the level of historical integrity, an issue for updates to dimension tables. When a dimension row is updated, related fact table rows are no longer historically accurate. For example, if the city column of a customer row changes, the related sales rows are no longer historically accurate. To preserve historical integrity, the related sales rows should point to an older version of the customer row. Kimball (April 1996) presents three alternatives for historical integrity:

- Type I: overwrite old values with the changed data. This method provides no historical integrity.
- Type II: use a version number to augment the primary key of a dimension table. For each change to a dimension row, insert a row in the dimension table with a larger version number. For example, to handle the change to the city column, there is a new row in the *Customer* table with the same customer number but a larger version number than the previous row. Besides the version number, additional columns are needed to record the beginning effective date and ending effective date for each historical column.
- Type III: use additional columns to maintain a fixed history. For example, to maintain a history of the current city and the two previous city changes, three city columns (*CustCityCurr*, *CustCityPrev*, *CustCityPast*) can be stored in the *Customer* table along with associated six date columns (two date columns per historical value column) to record the effective dates.

Figure 1.25 shows Type II and Type III alternatives for the *CustCity* column. The Type II alternative involves multiple rows for the same customer, but the entire history is represented. The Type III alternative involves just a single row for each customer, but only a limited history can be represented.

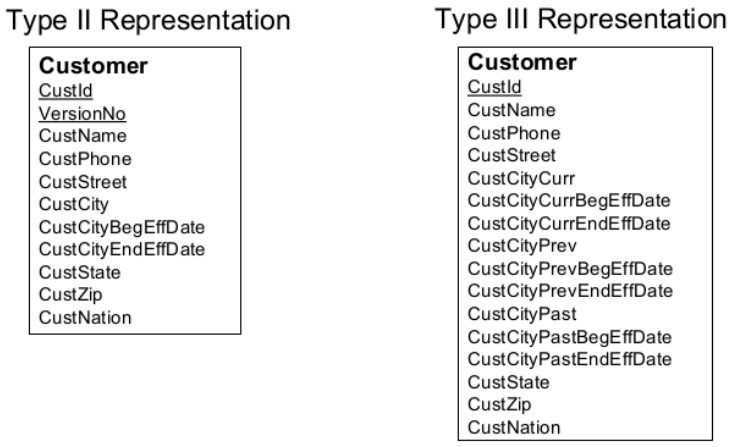


Figure 1.25: Alternatives for Historical Dimensional Integrity of *CustCity*

1.3.5 Dimension Representation

The star schema and its variations do not provide explicit representation of hierarchical dimensions because dimension tables do not define the hierarchical relationships among the levels of a dimension. Because dimension definition is important to support data cube operations as well as optimization techniques for query rewriting (presented in Chapter 17), many relational DBMS vendors have created proprietary SQL extensions for dimensions. This section reviews the Oracle CREATE DIMENSION statement to indicate the types of extensions that can be found in relational DBMSs.

The Oracle CREATE DIMENSION statement[1] supports the specification of levels, hierarchies, and constraints for a dimension. The first part of a dimension declaration involves the specification of levels. For flat (nonhierarchical) dimensions, there is only a single level in a dimension. However, most dimensions involve multiple levels as depicted in Example 1.1 for the *StoreDim* dimension. Each level corresponds to one column from the *Store* source table.

Example 1.1: Oracle CREATE DIMENSION Statement for the *StoreDim* Dimension with the Specification of Levels

```
CREATE DIMENSION StoreDim
 LEVEL StoreId    IS Store.StoreId
 LEVEL City       IS Store.StoreCity
 LEVEL State      IS Store.StoreState
 LEVEL Zip        IS Store.StoreZip
 LEVEL Nation     IS Store.StoreNation ;
```

The next part of a CREATE DIMENSION statement involves the specification of hierarchies. The Oracle CREATE DIMENSION statement supports dimensions with multiple hierarchies as shown in Example 1.2. Specification of a hierarchy proceeds from the most detailed level to the most general level. The CHILD OF keywords indicate the direct hierarchical relationships in a dimension.

Example 1.2: Oracle CREATE DIMENSION Statement for the *StoreDim* Dimension with the Specification of Levels and Hierarchies

```
CREATE DIMENSION StoreDim
 LEVEL StoreId    IS Store.StoreId
 LEVEL City       IS Store.StoreCity
 LEVEL State      IS Store.StoreState
 LEVEL Zip        IS Store.StoreZip
 LEVEL Nation     IS Store.StoreNation
 HIERARCHY CityRollup (
  StoreId CHILD OF
```

1 Do not put blank lines in CREATE DIMENSION statements. The Oracle SQL compiler generates error messages when encountering blank lines in CREATE DIMENSION statements.

```
  City    CHILD OF
  State   CHILD OF
  Nation )
HIERARCHY ZipRollup (
  StoreId CHILD OF
  Zip     CHILD OF
  State   CHILD OF
  Nation );
```

The Oracle CREATE DIMENSION statement supports dimensions with levels from multiple source tables. This feature applies to normalized dimension tables in snowflake schemas. Example 1.3 augments Example 1.2 with the inclusion of an additional level (*DivId*) along with an additional hierarchy containing the new level. In the level specification, the *DivId* level references the *Division* table. In the *DivisionRollup* hierarchy, the JOIN KEY clause indicates a join between the *Store* and the *Division* tables. The JOIN KEY clause at the end of the hierarchy specification is required when a hierarchy has levels from more than one source table.

Example 1.3: Oracle CREATE DIMENSION Statement for the *StoreDim* Dimension with the Usage of Multiple Source Tables

```
CREATE DIMENSION StoreDim
 LEVEL StoreId    IS Store.StoreId
 LEVEL City       IS Store.StoreCity
 LEVEL State      IS Store.StoreState
 LEVEL Zip        IS Store.StoreZip
 LEVEL Nation     IS Store.StoreNation
 LEVEL DivId      IS Division.DivId
 HIERARCHY CityRollup (
  StoreId CHILD OF
  City    CHILD OF
  State   CHILD OF
  Nation )
HIERARCHY ZipRollup (
  StoreId CHILD OF
  Zip     CHILD OF
  State   CHILD OF
  Nation )
HIERARCHY DivisionRollup (
  StoreId CHILD OF
  DivId
  JOIN KEY Store.DivId REFERENCES DivId );
```

The final part of a CREATE DIMENSION statement involves the specification of constraints. The ATTRIBUTE clause defines functional dependency relationships involving dimension levels and nonsource columns in dimension tables. Example 1.4 shows ATTRIBUTE clauses for the non source columns in the *Division* table.

Example 1.4: Oracle CREATE DIMENSION Statement for the *StoreDim* Dimension with the Usage of ATTRIBUTE Clauses for Constraints

```
CREATE DIMENSION StoreDim
 LEVEL StoreId    IS Store.StoreId
 LEVEL City       IS Store.StoreCity
 LEVEL State      IS Store.StoreState
 LEVEL Zip        IS Store.StoreZip
 LEVEL Nation     IS Store.StoreNation
 LEVEL DivId      IS Division.DivId
 HIERARCHY CityRollup (
```

```
  StoreId CHILD OF
  City    CHILD OF
  State   CHILD OF
  Nation )
HIERARCHY ZipRollup (
  StoreId CHILD OF
  Zip     CHILD OF
  State   CHILD OF
  Nation )
HIERARCHY DivisionRollup (
  StoreId CHILD OF
  DivId
  JOIN KEY Store.DivId REFERENCES DivId )
ATTRIBUTE DivId DETERMINES Division.DivName
ATTRIBUTE DivId DETERMINES Division.DivManager ;
```

In Example 1.4, the DETERMINES clauses are redundant with the primary key constraint for the *Division* table. The DETERMINES clauses are shown in Example 1.4 to reinforce the constraints supported by the primary key declarations. DETERMINES clauses are required for constraints not corresponding to primary key constraints to enable query optimizations. For example, if each zip code is associated with one state, a DETERMINES clause should be used to enable optimizations involving the zip and state columns.

1.4 Enterprise Data Warehouse Development

Enterprise data warehouse development requires a methodology to apply the principles and practices of data modeling and data integration. Without an appropriate methodology, the best principles and practices will likely fail to produce a data warehouse with high value for an organization. The first subsection reviews a range of proposed methodologies to provide background and insights for successful development of enterprise data warehouses. To solidify understanding of the important concepts in this chapter, the second subsection presents the Colorado Education Data Warehouse, a moderate-size data warehouse that depicts major concepts presented in this chapter.

1.4.1 Data Warehouse Design Methodologies

Design methodologies for data warehouses differ from methodologies for transaction databases because data warehouses are primarily repositories of secondary data. Operational databases capture data at its source such as from an ATM, web shopping cart, and manufacturing plant. In contrast, data warehouses transform data from primary sources (operational databases and external data sources) and then store and summarize the transformed data.

Because of differences in data characteristics and usage, data warehouses have different design artifacts than operational databases. Data warehouse design methodologies support the development of data models, data integration procedures, and data marts for enterprise data warehouses. The data models for data warehouses have different patterns than data models for operational databases. Data integration procedures are essential for data warehouses but typically not important for operational databases. Data marts have different characteristics than views for operational databases.

This section presents several methodologies for data warehouse design. These methodologies have received some prominence although numerous other approaches also have been proposed. Data warehouse design methodologies differ by emphasis on the supply of data sources, the demand for business intelligence services, and the possible level of automation in the development process. Automation may have an important role because data sources already exist as raw materials for data warehouse design. However, no commercial products have been developed to support automation in data warehouse design methodologies.

Demand Driven Methodology

The demand-driven data warehouse design methodology (also known as the requirements driven approach), first proposed by Kimball et al. (1998), is one of the earliest data warehouse design methodologies. The demand-driven methodology emphasizes the identification of data marts to capture intended usage of a data warehouse as depicted in

Figure 1.26. A data mart is defined as a collection of related facts important for a group of data warehouse users. When this methodology was proposed, the data mart architecture was common for data warehouses.

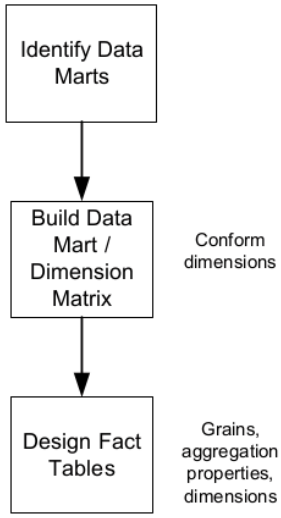


Figure 1.26: Steps in the Demand-Driven Data Warehouse Design Methodology

After identifying data marts, possible dimensions for each data mart are listed. Dimensions, standardized across data marts, are known as conformed dimensions. A matrix relating conformed dimensions and data marts is developed to refine the initial data mart specification.

The final step involves specification of fact tables with an emphasis on the grain of fact tables. Typical grains are individual transactions, snapshots (points in time), and line items on documents. The grain is usually determined by the primary dimensions. After specifying the grain of a fact table, all dimensions are specified. The details of each dimension are then specified including the hierarchical levels. In the last part, measures for each fact table are specified including the measure properties such as aggregation.

Demand-driven data warehouse design methodology: emphasizes the identification of data marts to capture intended usage of a data warehouse. The demand-driven methodology has three phases for (1) identifying data marts (subsets of user requirements), (2) building a matrix relating data marts and dimensions, and (3) designing fact tables.

Supply Driven Methodology

The supply-driven data warehouse design methodology (Moody and Kortink, 2000) emphasizes the analysis of existing data sources. Entity types in ERDs of existing data sources are analyzed to provide a starting point for the data warehouse design as shown in Figure 1.27. The supply-driven methodology seems amenable to automation although automated tools to support the methodology have not been reported.

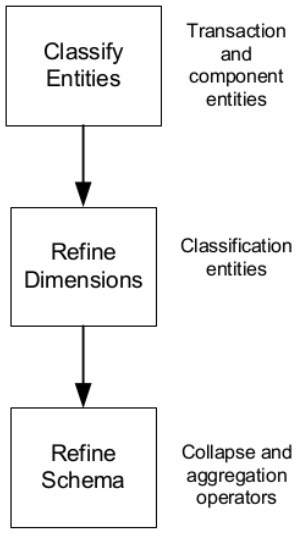


Figure 1.27: Steps in the Supply-Driven Data Warehouse Design Methodology

In the first step, the supply-driven approach classifies entity types in existing ERDs as a prelude to develop a star schema or variation for the data warehouse. Entity types containing event data at a point in time are classified as transaction entity types. Guidelines in the methodology indicate that transaction entity types typically have numeric data that can be summarized. Typical events involve sales, purchases, reservations, hiring, and so on. Event entity types will typically become fact tables in a star schema. Entity types related to events in 1-M relationships are classified as component entity types. Component entity types typically become dimension tables in a star schema. In total, the first step provides a set of initial star schemas or a constellation schema if dimensions are conformed.

The second step of the supply-driven methodology refines dimensions. Entity types related to component entity types are labeled as classification entity types. Dimension hierarchies are formed by classification and component entity types. Each sequence of classification and component entity types, joined by 1-M relationships in the same direction, becomes a dimension hierarchy.

The third step of the methodology refines the star schemas using two operators. The collapse operator denormalizes dimension entity types to reduce snowflaking. For example, the collapse operator applied to the Division-Store relationship in Figure 1.11 combines the *Division* and *Store* entity types eliminating the snowflake design. The aggregation operator makes the grain coarser in transaction entity types. Aggregation of a fact table may require modifications to the primary dimension tables to make the dimension tables consistent with the grain of the fact table.

Supply-driven data warehouse design methodology: emphasizes the analysis of existing data sources. Entities in ERDs of existing data sources are analyzed to provide a starting point for the data warehouse design. The supply-driven methodology has three phases for (1) classify entities, (2) refine dimensions, and (3) refine the schema.

Hybrid Methodology

The hybrid data warehouse design methodology (Bonifati et al., 2001) combines the demand and supply methodologies. The hybrid methodology involves a demand-driven stage, a supply-driven stage, and then a third stage to integrate the demand and supply-driven stages. The demand and supply stages can be done independently as shown in Figure 1.28. The overall emphasis in the hybrid approach is to balance the demand and supply aspects of data warehouse design possibly aided by automated tools.

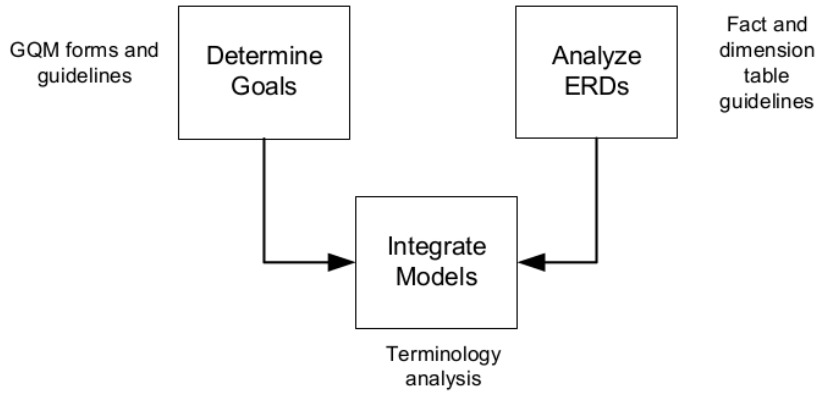


Figure 1.28: Steps in the Hybrid Data Warehouse Design Methodology

The demand-driven stage collects requirements using the Goal-Question-Metrics (GQM) paradigm. The GQM provides forms and interview guidelines to help define a set of goals for the data warehouse. The methodology provides some informal guidelines to derive measures and dimensions from the goals.

The second step of the hybrid methodology involves analysis of existing ERDs. This step may be partially automated although tool development has not been reported. The methodology provides guidelines to identify fact and dimension tables in existing ERDs. Potential fact tables are identified based on the number of additive attributes. Dimension tables are involved in 1-M relationships with fact tables.

The third step of the hybrid methodology integrates the dimensional model in the demand stage and the star schema in the supply stage. The methodology provides guidelines to convert both models to a common vocabulary using terminology analysis. After conversion to a common vocabulary, the methodology provides a process to match the demand and supply models.

Hybrid data warehouse design methodology: combines the demand and supply methodologies. The hybrid methodology involves a demand-driven stage, a supply-driven stage, and then a third stage to integrate the demand and supply-driven stages. The demand and supply stages can be done independently. The overall emphasis in the hybrid approach is to balance the demand and supply aspects of data warehouse design.

1.4.2 Colorado Education Data Warehouse

The Colorado Education Data Warehouse supports reporting of student assessments and growth in K-12 schools in Colorado. Assessments of student achievement provide evidence of the current status of student knowledge and understanding. However, measurement of learning requires assessment of growth in achievement over time, not just the current status of knowledge. Since learning is the primary responsibility of education, Colorado and other states have invested in systems to support assessment of student growth. In Colorado, a 1997 law required the development of accountability systems for K-12 education. This law was extended in 2004 and 2007 to measurement of student growth, not just student achievement at a point in time.

As a response to the 2007 law, the Colorado Department of Education extended its Education Data Warehouse and developed web portals to support assessments of student growth. The original Education Data Warehouse was developed beginning in 2002 to support education accountability. The student growth extension, performed from 2007 to 2009, had a budget of $6.7 million. As a result of the development since 2002, the Education Data Warehouse is in a mature state with data governance policies, processes and standards to manage the flow of data from capture to use. Data stewards provide data quality audits as part of the ongoing monitoring of data quality facilitated by master data management technology.

The Colorado Department of Education has embarked on two major expansion projects in 2013. The Data Warehouse Expansion Project will add data from preschool to career and extend security and accuracy in teacher and student data reporting. The Data Pipeline Project supports efficient transfer of data from school districts to the education data warehouse. The Data Pipeline will reduce data redundancy, capture data with less time lag, create transaction interchanges to streamline the data collection process, and support exchange of data on transferred students.

Colorado SchoolView™ (http://www.schoolview.org) is a public portal that uses the Education Data Warehouse. SchoolView supports visual analysis of student growth on the Colorado Student Assessment Program (CSAP) tests for all Colorado school districts. Users can compare median student growth in reading, writing, and math by school as depicted in Figure 1.29. In addition, users can search on the dimensions of student group, grade, and ethnicity as well as rolling up to school districts. Figure 1.30 displays visual results for the student group dimension. By selecting a bubble in the display, users can drill down to the individual member value. SchoolView provides a map interface in addition to standard searching tools for selecting schools and school districts.

Figure 1.29: SchoolView Window with Visual Display of Math Results for Individual Schools

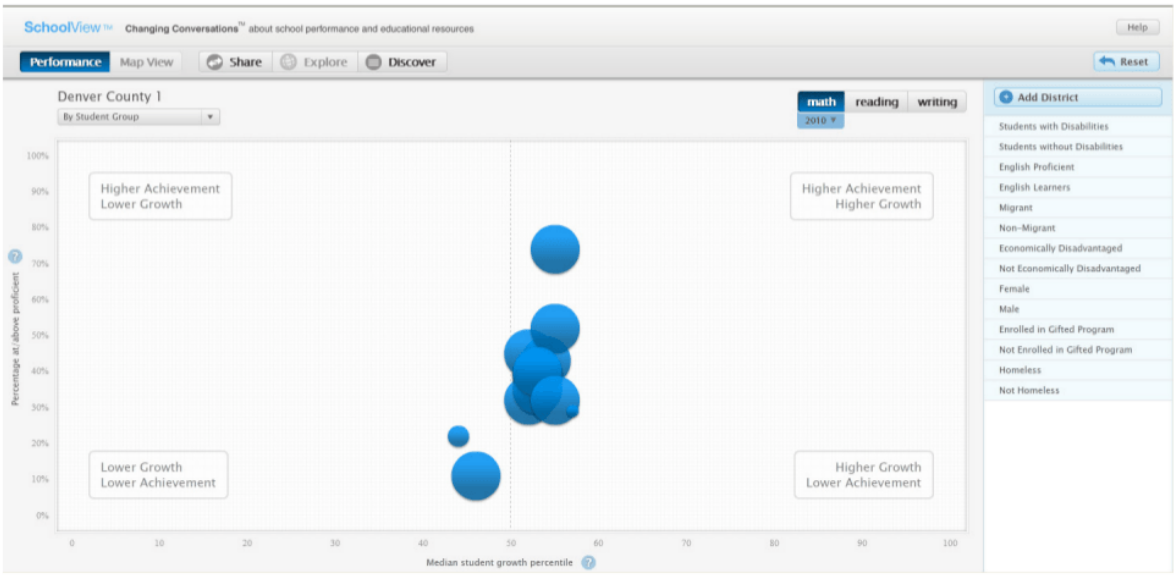


Figure 1.30: SchoolView Window with Visual Display of Math Results for Student Groups

The SchoolView portal serves parents, teachers, school administrators, and government agencies. Parents use the portal as a decision aid for school enrollment of their children. Colorado permits parental choice in enrollment although school district boundaries may restrict the choice somewhat. School boards at the school district level use SchoolView to make decisions about resource allocations to individual schools and possible remedial actions for individual schools. Teachers and school administrators need aggregate and individual student data to decide on program effectiveness and student performance. Government agencies use SchoolView in decisions about education policy. In particular, the U.S. federal "No Child Left Behind" law has requirements fulfilled by SchoolView. The public part of SchoolView serves parents and the general public. Educators and government agencies have access to the private part of SchoolView.

The primary data source for the data warehouse is the CSAP scores. Students take CSAP tests once per year in the second half of the school year. CSAP scores are maintained by the Colorado Department of Education independent of usage in SchoolView. School districts provide data for student grades and demographic attributes. Colorado has 178 school districts so the number of data sources provided by school districts is large. Figure 1.31 shows other data sources provided by higher education, public safety, corrections, early childhood development, and human services.

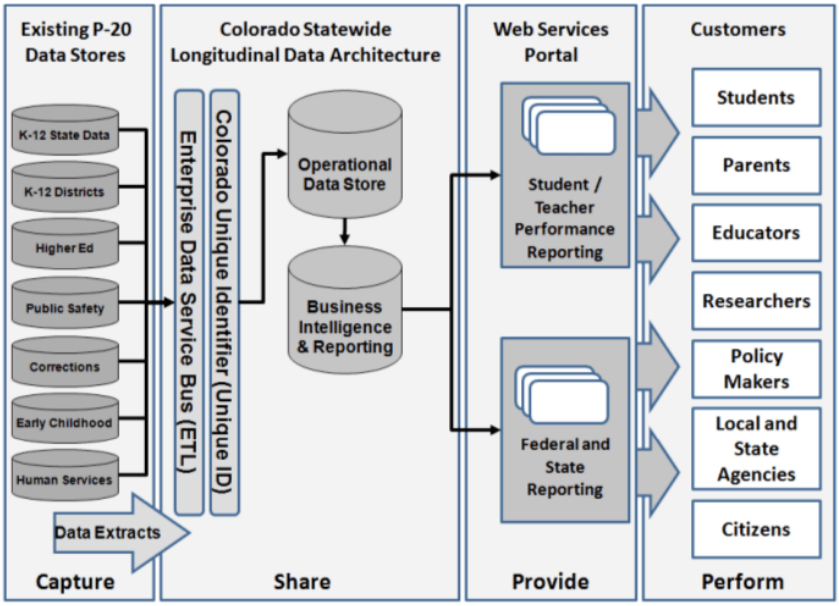


Figure 1.31: Enterprise Architecture for the Extended Education Data Warehouse

Source: 2009 Colorado SLDS Application Grant Proposal

Several important innovations have been made in the data integration process. The most important innovation was the development of state wide identifiers for teachers and students. Due to the importance of tracking students and teachers across time and schools, state wide identifiers were essential. To facilitate movement of application data sources, a data exchange portal was developed. This portal supports transformation of data about staff, finance, school districts, safety, discipline, and student enrollment. To support transformation of test and growth results, an assessment portal was developed. Despite development of these portals, a large number of dimension tables (about 30) must be manually processed. Some of the manually processed dimension tables are small and stable (such as ethnicity, gender, and grade), while others are reasonably large and somewhat dynamic (such as facilities and job classes).

The data model for the Education Data Warehouse is a complex collection of nine constellation schemas. The constellation schemas contain 94 dimension tables and 32 fact tables. The fact tables have some level of denormalization as the fact tables contain both key and code values instead of just the key values. The constellation schemas have some level of snowflaking as ten dimension tables are referenced in other dimension tables. The schema diagram uses nine pages in Microsoft Visio with each page containing a constellation schema with many connections among fact and dimension tables.

One of the simplest schemas contains the fact table for student CSAP scores along with connections to 27 dimension tables as shown in Table 1-11. The dimension tables demonstrate most of the concepts in earlier parts of this chapter. Most dimension tables contain flat dimensions, typically a code value. For example, dimension tables for ethnicity, homeless, migrant status, and gifted contain a single, flat dimension. Some dimension tables contain hierarchical dimensions and multiple dimensions. For example, the school dimension has location columns that form hierarchical dimensions and other columns comprising flat dimensions. Each dimension table uses two date columns (beginning and ending effective dates) to provide historical integrity.

Table 1-11: List of Dimensions in the Student CSAP Star Schema

DIM_504_PLAN	DIM_ESL	DIM_IEP
DIM_ACCOMMODATION	DIM_ETHNICITY	DIM_LANGUAGE_BACKGROUND
DIM_BILINGUAL	DIM_FARM	DIM_MIGRANT_STATUS
DIM_CBLA_STATUS_CODE	DIM_GENDER	DIM_SCHOOL_EMH
DIM_CSAP_CONTENT_PROFICIENCY	DIM_GIFTED_TALENTED	DIM_SCHOOL_YEAR
DIM_CSAP_SUBJECT	DIM_GRAD_CLASS	DIM_SCHOOL
DIM_DID_NOT_TEST	DIM_GRADE_CALC_EXEMPTION	DIM_TIME_IN_DISTRICT
DIM_DISABLING_CONDITION	DIM_GRADE	DIM_TIME_IN_SCHOOL
DIM_DISTRICT	DIM_HOMELESS	DIM_TITLE_1

The fact tables contain measures with a variety of aggregation properties. The major fact table for CSAP results contains additive measures (number of points, percentage points, scaled score, and growth percentile) although these measures should be shown as central tendencies (average or median) for reasonable user interpretation. Summary fact tables contain event counts. For example, the CSAP summary fact table contains counts of the types of test results such as the number of partially proficient students. The finance fact tables contain semi-additive measures that can be summarized across time such as enrollment counts and bonded debt levels.

The Education Data Warehouse is modest size compared to enterprise data warehouses of major corporations. The total size of the Education Data Warehouse is about 270 GB. The fact tables have about 200 million rows with 1.6 million rows added per year to the major CSAP fact table.

This background on the Education Data Warehouse should provide insight about the difficulty to develop and operate even a moderate-size data warehouse. The Education Data Warehouse has evolved over 10 years of development with initial development in 2002, followed by a major extension since 2007, along with plans for even larger extensions. The development effort required coordination among many areas of state and local government in Colorado. The operation of the data warehouse involved many new policies for data quality. The effort resulted in a complex product, much too large and complex to show in this textbook.

Closing Thoughts

This chapter provided a detailed introduction to the concepts and design practices of data warehousing. The chapter began by examining conceptual differences between relational databases, traditionally used for transaction processing, and multidimensional databases, suggested for the new generation of business intelligence applications. The unique characteristics of business intelligence data were outlined, followed by a discussion of data warehouse architectures, data mining, and data warehouse usage in organizations.

Data warehouses can be implemented using the multidimensional data model, the relational model, or a combination of both models. For the multidimensional data model, this chapter presented the terminology associated with data cubes and the operators to manipulate data cubes. For the relational data model, the chapter presented data modeling techniques (the star schema and its variations), design rules for summarizability, time representation in relational data warehouse designs, and extensions for representation of hierarchical dimensions.

The final part of the chapter emphasized development of enterprise data warehouses. Several prominent methodologies for designing enterprise data warehouses were reviewed. To depict the complexities of enterprise data warehouse development, the Colorado Education Data Warehouse was presented.

Review Concepts

- Data needs for transaction processing versus business intelligence applications
- Characteristics of a data warehouse: subject-oriented, integrated, time-variant, and nonvolatile
- Architectures for deploying a data warehouse: two-tier, three-tier, bottom-up
- Staging area to provide temporary storage of transformed data before loading into a data warehouse
- Stages of the Data Warehouse Maturity Model (infant, child, teenager, adult, and sage) and difficulty of moving between some stages (infant to child and teenager to adult)
- Multidimensional data cube: dimensions, measures, hierarchies, time-series data type
- Important dimension properties: hierarchy and sparsity
- Aggregation property for measures: additive, semi-additive, non-additive
- Data cube operators: slice, dice, drill-down, roll-up, pivot
- Star schema: fact table and related dimension tables
- Variations of the star schema: snowflake schema (multiple dimension levels) and constellation schema (multiple fact tables and shared dimension tables)
- Classifications of fact tables: transaction table, snapshot table, and factless table
- Dimension summarizability problems: drill-down incompleteness, roll-up incompleteness, and non strict dimensions
- Dimension summarizability patterns: regular dimension pattern and unusual dimension pattern
- Fact-dimension summarizability problems: incomplete dimension-fact relationships and non strict dimension-fact relationships
- Fact-dimension summarizability patterns: regular dimension-fact pattern and unusual dimension-fact pattern
- Maintaining historical dimensional integrity using a Type II representation for unlimited history and a Type III representation for limited history
- Dimension representation to support data cube operations and optimization techniques
- Demand-driven data warehouse design methodology emphasizing the identification of data marts to capture intended usage of a data warehouse
- Supply-driven data warehouse design methodology emphasizing the analysis of existing data sources
- Hybrid data warehouse design methodology balancing the demand and supply-driven methodologies

Questions

1. Why are operational databases not particularly suited for business intelligence applications?
2. How is a data warehouse different from a data mart?
3. When is the three-tier data warehouse architecture more appropriate than the two-tier data warehouse architecture?
4. What are the components of an enterprise data model?

5. What are some causes for failures in data warehouse projects?
6. Does a bottom-up data warehouse architecture use an enterprise data model?
7. What is an oper mart?
8. What is the purpose of the data warehouse maturity model?
9. What is an important insight provided by the data warehouse maturity model?
10. What are the advantages of multidimensional representation over relational representation for business analysts?
11. Explain why a dimension may have multiple hierarchies.
12. Why is it important to specify the aggregation property for each measure?
13. Provide an example of a measure with each aggregation property value (additive, semi-additive, and non-additive).
14. What are the advantages of using time-series data in a cell instead of time as a dimension?
15. How is slicing a data cube different from dicing?
16. What are the differences between drilling-down and rolling-up a data cube dimension?
17. How is a pivot operation useful for multidimensional databases?
18. Explain the significance of sparsity in a data cube.
19. What is a star schema?
20. What are the differences between fact tables and dimension tables?
21. How does a snowflake schema differ from a star schema?
22. What is a constellation schema?
23. What is the relationship between the types of fact tables and aggregation properties for measures?
24. What is the difference between drill-down incompleteness and roll-up incompleteness?
25. For the non strict dimension problem, will totals summarized at the child level always be larger than totals summarized in the related parent level?
26. Why is the unusual dimension summarizability pattern called unusual?
27. List the distinguishing schema patterns for the regular summarizability dimension pattern, the drill-down incomplete pattern, and the roll-up imcomplete pattern.
28. Briefly explain the incomplete dimension fact relationship problem including the schema pattern for the problem.
29. Briefly explain the difference between the non strict dimension problem and the non strict dimension-fact relationship problem.
30. What is the difference between the regular and unusual patterns for summarizable dimension-fact relationships?
31. Briefly explain two ways to resolve non strict dimension-fact relationships.
32. Briefly explain resolution of the incompleteness dimension summarizability problems.
33. Briefly explain resolution of the incomplete dimension-fact relationship problem.

34. How is time represented in a fact table?
35. What is an accumulating fact table?
36. What is the difference between Type II and Type III representations for historical dimension integrity?
37. What is the purpose of the Oracle CREATE DIMENSION statement?
38. Why are new design methodologies needed for data warehouse design instead of using approaches for design of operational databases?
39. Briefly explain the demand-driven data warehouse design methodology.
40. Briefly explain the supply-driven data warehouse design methodology.
41. Briefly explain the hybrid data warehouse design methodology.

Problems

The problems provide practice with data cube definition and operations as well as star schema design. The problems involve the database used by an automobile insurance provider (Figure 1.P1) to support policy transactions (create and maintain customer policies) and claims transactions (claims made by other parties). This database design is simplified to provide reasonable practice problems with the concepts in Chapter 1. The policy transactions utilize the entity types *Item*, *Agent*, *InsuredParty*, *Policy*, *InsuredAuto*, and *PolicyItem*, while the claims transactions use the entity types *InsuredParty*, *Claimant*, *InsuredAuto*, *Policy*, and *Claim*. For each entity type, you should assume the data types of your own choice. The cardinalities for the *InsuresParty* relationship indicate that a policy can involve an entire family, not just individuals.

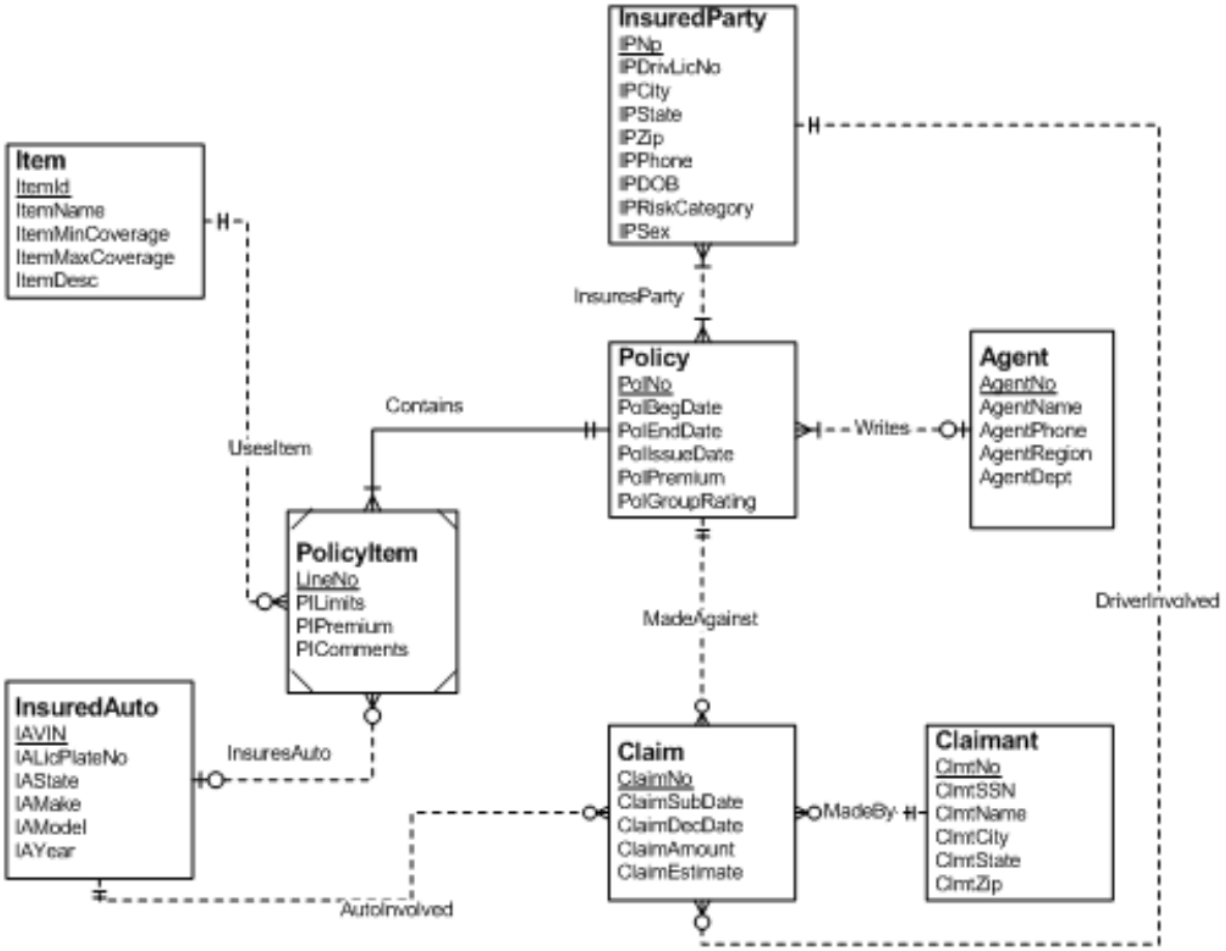


Figure 1.P1: ERD for Auto Insurance Policies and Claims

1. Identify dimensions and measures in a data cube for automobile policy analysis. Indicate the aggregation property of each measure.

2. Identify the finest level grain of the fact table to support automobile policy analysis. Justify your decision.

3. Consider a relationship between a dimension for insured parties and the fact table in problem 2. Identify a summarizability problem involving this relationship. Propose an alternative to resolve the summarizability problem.

4. Consider a relationship between a dimension for agents and the fact table in problem 2. Identify a summarizability problem involving this relationship. Propose an alternative to resolve the problem.

5. Consider relationships between dimension tables for items and insured autos and the fact table in problem 2. Identify any summarizability problems in the relationships. Propose an alternative representation for each summarizability problem that you identified.

6. Design a time dimension table and one or more relationships to the fact table in problem 2.

7. Design a star or snowflake schema to support the dimensions and measures in the data cube from problem 1. Your schema should use resolutions to summarizability problems that you proposed in earlier problems.

8. For each dimension table in the schema for problem 7, list the independent and hierarchical dimensions. Analyze each hierarchical dimension for summarizability problems. Propose alternative representations to resolve any

summarizability problems.

9. Identify dimensions and measures in a data cube for claims analysis. Indicate the aggregation property of each measure.

10. Identify the finest level grain of the fact table to support automobile claim analysis. Justify your decision.

11. Consider a relationship between a dimension for claimants and the fact table in problem 10. Identify any summarizability problems involving this relationship. Propose an alternative to resolve the problem if a problem exists.

12. Consider a relationship between a dimension for insured autos and the fact table in problem 10. Identify any summarizability problems involving this relationship. Propose an alternative to resolve the problem if a problem exists.

13. Consider a relationship between a dimension for policies and the fact table in problem 10. Identify any summarizability problems involving this relationship. Propose an alternative to resolve the problem if a problem exists.

14. Consider a relationship between a dimension for insured party and the fact table in problem 10. Identify any summarizability problems involving this relationship. Propose an alternative to resolve the problem if a problem exists.

15. Design a time dimension table and one or more relationships to the fact table in problem 10.

16. Design a star or snowflake schema to support the dimensions and measures in the data cube for problem 9. Your schema should use resolutions to summarizability problems that you proposed in earlier problems.

17. For each dimension table in the schema for problem 16, list the independent and hierarchical dimensions. Analyze each hierarchical dimension for summarizability problems. Propose alternative representations to resolve any summarizability problems.

18. Combine the star or snowflake schemas for policies (problem 7) and claims (problem 16) into a constellation schema. Identify common dimension tables that can be shared in the constellation schema.

19. For the *InsuredPartyDim* table, discuss the stability of the columns in the table. What columns would typically change together? What columns would history be desirable?

20. For the *InsuredAutoDim* table, discuss the stability of the columns in the table. What columns would typically change together? What columns would history be desirable?

21. Modify the *InsuredPartyDim* table for a history of the *IPRiskCategory* column. Provide a type II representation and a type III representation with current and previous risk values.

22. Modify the *InsuredPartyDim* table for a limited history of the *IPCity*, *IPState*, and *IPZip* columns. The limited history should record the current and previous values and change dates for the combination of columns.

23. Describe the data cube resulting from the operation to slice the policy data cube by a certain agent.

24. Describe the data cube resulting from the operation to dice the data cube result of the slice operation in problem 9 by insured parties having zip codes in a specified state.

25. Begin with a data cube with four dimensions (*InsuredParty*, *InsuredAuto*, *Item*, and *Agent*) and one measure (policy amount) in the cells. From this data cube, describe the operation to generate a new data cube with three dimensions (*InsuredParty*, *Item*, and *Agent*) and one measure (average auto policy amount).

26. This problem involves the addition of external telematics data into the auto insurance data warehouse design. Telematics involves advanced sensors installed in a car or a smartphone to collect data on an individual car usage including time, location, types of roads, and driving behavior such as speed. The external data source is an XML file with the following structure. The XML file only contains data for drivers insured by the auto insurance company. For this problem, identify the dimensions and measures involved in the XML file. Think carefully about the grain for the measures as the XML file is very large with time intervals every 5 to 10 minutes per driving

experience.

- Header record: VIN, license plate number, state, driver license number, insurance policy number.
- Detail record (multiple records for the header record): date, start time, end time, start GPS coordinates, end GPS coordinates, driving distance, road type, average driving speed, and average speed limit.Design a star or snowflake schema to support the dimensions and measures in the data cube from problem 26. Utilize the dimensions in the schemas from problem 18 as much as possible. You can assume that the data integration process would add data missing from the XML file for some dimensions.

27. Problems 28 to 30 involve a data warehouse schema to integrate the data sources shown in Figure 1.P2 and Tables 1-P1 to 1-P5. The ERD in Figure 1.P2 supports the purchasing operation of a moderate size retailer with one preferred supplier per product. Tables 1-P1 to 1-P4 show sample data for the tables in the purchases database. The supply purchase spreadsheet (Table 1-P5) contains purchases of small amounts to support office operations. In this problem, you should identify all relevant measures and dimensions with hierarchies specified to support inventory analysis.

28. Extend your analysis to design a star schema (or variation) to support inventory analysis. You should identify summarizability problems in your star schema and indicate mapping from data sources into tables.

29. You should populate your data warehouse tables based on the data in the operational tables and spreadsheet. You do not need to insert the data into your tables. You can just show table listings in your solution. You should indicate mapping from data sources into tables.

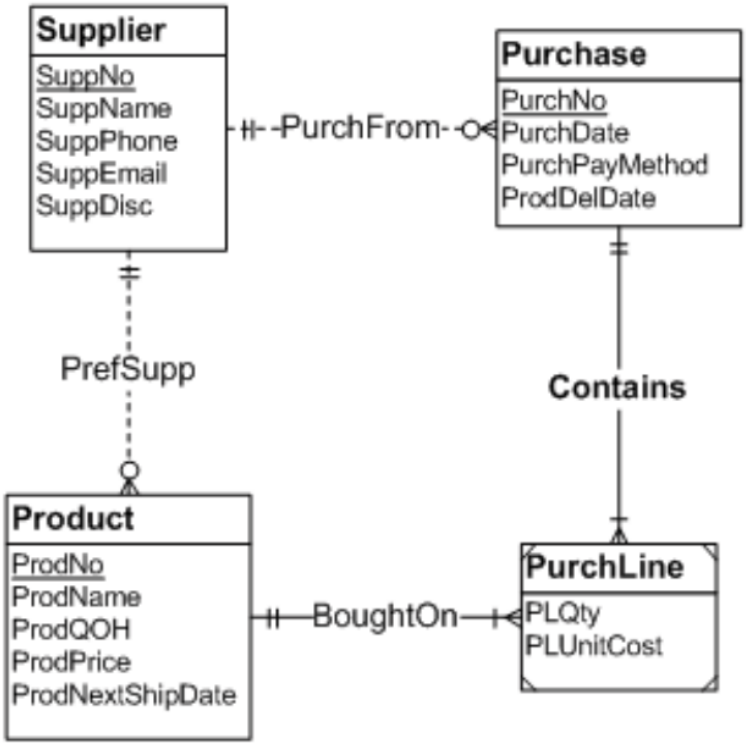


Figure 1.P2: ERD for Retail Purchase Operations

Table 1-P1: Sample Data for the *Supplier* Table

SuppNo	SuppName	SuppEmail	SuppPhone	SuppDisc
S2029929	ColorMeg, Inc.	custrel@colormeg.com	(720) 444-1231	0.10
S3399214	Connex	help@connex.com	(206) 432-1142	0.12
S4290202	Ethlite	ordering@ethlite.com	(303) 213-2234	0.05
S4298800	Intersafe	orderdesk@intersafe.com	(512) 443-2215	0.10
S4420948	UV Components	custserv@uvcomponents.com	(303) 321-0432	0.08
S5095332	Cybercx	orderhelp@cybercx.com	(212) 324-5683	0.00

Table 1-P2: Sample Data for the *Product* Table

ProdNo	ProdName	SuppNo	ProdQOH	ProdPrice	ProdNextShipDate
P0036566	17 inch Color Monitor	S2029929	12	$169.00	02/20/2013
P0036577	19 inch Color Monitor	S2029929	10	$319.00	02/20/2013
P1114590	R3000 Color Laser Printer	S3399214	5	$699.00	01/22/2013
P1412138	10 Foot Printer Cable	S4290202	100	$12.00	
P1445671	8-Outlet Surge Protector	S4298800	33	$14.99	
P1556678	CVP Ink Jet Color Printer	S3399214	8	$99.00	01/22/2013
P3455443	Color Ink Jet Cartridge	S3399214	24	$38.00	01/22/2013
P4200344	36-Bit Color Scanner	S4420948	16	$199.99	01/29/2013
P6677900	Black Ink Jet Cartridge	S3399214	44	$25.69	
P9995676	Battery Back-up System	S5095332	12	$89.00	02/01/2013

Table 1-P3: Sample Data for the *Purchase* Table

PurchNo	PurchDate	SuppNo	PurchPay-Method	PurchDelDate
P2224040	02/03/2013	S2029929	Credit	02/08/2013
P2345877	02/03/2013	S5095332	PO	02/11/2013
P3249952	02/04/2013	S3399214	PO	02/09/2013
P3854432	02/03/2013	S4290202	PO	02/08/2013
P9855443	02/07/2013	S4420948	PO	02/15/2013

Table 1-P4: Sample Data for the *PurchLine* Table

PurchNo	ProdNo	PLQty	PLUnitCost
P2224040	P0036566	10	$100.00
P2224040	P0036577	10	$200.00
P2345877	P9995676	10	$45.00
P3249952	P1114590	15	$450.00
P3249952	P1556678	10	$50.00
P3249952	P3455443	25	$21.95
P3249952	P6677900	25	$12.50
P3854432	P1412138	50	$6.50
P9855443	P4200344	15	$99.00

Table 1-P5: Sample Spreadsheet Data for Supply Purchases

ProdCode	ProdDesc	Supp	Qty	Unit Price	PurchDate	Amount
CPC1	No 2 pencils	Omart	20	$2.00	13-Feb-2013	$40.00
CPC2	Copier paper	Smart	10	$3.50	14-Feb-2013	$35.00
CPC3	File folders	Pmart	20	$1.50	11-Feb-2013	$30.00

References for Further Study

Several references provide additional details about important parts of Chapter 1. The survey of summarizability concepts by Mazon et al. (2009) augments Sections 1.3.2 and 1.3.3 on summarizability problems and patterns. The survey of data warehouse design methodologies by Romero and Abello (2009) augments Section 1.4.1 on design methodologies. More details about the Data Warehouse Maturity Model can be found in Eckerson (2007) while the TDWI website (tdwi.org) provides details about the Business Intelligence Maturity Model. Kimball (1996, 2003) provides more details about historical integrity covered in Section 1.3.4.

Chapter 2

Data Integration Practices and Relational DBMS Extensions for Data Warehouses

Learning Objectives

This chapter extends the foundation for data warehouses provided in Chapter 1 with details about data integration and relational DBMS extensions. After this chapter, the student should have acquired the following knowledge and skills:

- Explain the types of data sources, data quality issues, and data cleaning tasks involved in data integration
- Understand the features of integration tools for maintaining a data warehouse
- Gain insight about managing the complex processes of refreshing and populating data warehouses
- Write SQL SELECT statements using the CUBE, ROLLUP, and GROUPING SETS operators
- Understand conceptual differences between materialized views for summary data storage and retrieval and traditional relational views for retrieval of derived data
- Gain insights about the complexity of the query rewriting process to match materialized views with user queries

Overview

Chapter 1 provided conceptual background and data modeling skills as a foundation for understanding data warehouse development in organizations. You should have a clear understanding about the differences between data warehouses and operational databases, architectures to support data warehouse development, multidimensional representation for business analysts, data modeling practices, and data warehouse design methodologies. Chapter 2 extends this foundation with details about data integration, query language extensions for multidimensional data, and efficient management of summarized data. Together, Chapters 1 and 2 provide skills and knowledge essential to careers and work assignments involving data warehouses.

Chapter 2 emphasizes the operation, usage, and physical implementation of data warehouses. You will first learn about data integration concepts and practices. Chapter 2 covers characteristics of data sources, workflow specification of data integration tasks, details of data cleaning tasks, data integration tools, and management of complex refresh processes. Next you will learn about query language extensions for data warehouse retrievals. You will learn details about extensions to the SQL GROUP BY clause for calculating subtotals in query results. In the last part of Chapter 2, you will learn about extensions to relational DBMSs for efficient storage and retrieval of summary data. You will learn about materialized views, query rewriting principles to substitute materialized views in user queries, optimization techniques employed by relational DBMSs, and architectures to support data warehouse processing. Since relational

DBMSs provide the underlying storage and retrieval capabilities for enterprise data warehouses, understanding relational DBMS features for data warehouses is an essential part of a student's background.

2.1 Data Integration Concepts

Data integration adds value to disparate data sources that contribute data in enterprise data warehouses. Although data warehouses largely contain replicated data, populating and maintaining a data warehouse is much more complex than copying source data. Integrating data sources involves challenges of large volumes of data, widely varying formats and units of measure, different update frequencies, missing data, and lack of common identifiers.

Data integration facilitates initially populating a data warehouse and periodically refreshing a data warehouse as data sources change. Determining data to load in a warehouse involves matching business intelligence needs to the realities of available data. Reconciling the differences among data sources is a significant challenge especially considering that source systems typically cannot be changed. As data sources change, a data warehouse should be refreshed in a timely manner to support business intelligence needs. Since data sources change at different rates, the determination of the time and content to refresh can be a significant challenge. Because of these challenges, data integration can involve significant investments in hardware, software, and personnel to achieve a satisfactory solution.

This section presents concepts and practices of data integration. The first part describes the kinds of data sources available for populating a data warehouse. The second part describes workflow specification for maintaining a data warehouse. The third part delineates details of data cleaning tasks, essential elements in workflows for maintaining a data warehouse. The fourth part presents data integration architectures and reviews features of commercial data integration tools. The final part discusses the problem of determining the frequency of refreshment and the content to refresh.

2.1.1 Sources of Data

Accessing source data presents challenges in dealing with a variety of formats and constraints on source systems. External source systems usually cannot be changed. Internal source systems may or may not be changeable to accommodate the requirements of a data warehouse. Even if a source system can be changed, budget constraints may allow only minor changes. Source data may be stored in legacy format or modern format. Legacy format generally precludes retrieval using nonprocedural languages such as SQL. Modern format means that the source data can be accessed through a relational database or Web pages. Unless stored with formal meta data, Web pages can be difficult to parse and nonstandard across websites. Formal meta data usually involves XML data along with an XML schema to provide interpretation of the XML data.

Change data from source systems provides the basis to update a data warehouse. Change data comprises new source data (insertions) and modifications to existing source data (updates and deletions). Further, change data can affect fact tables and/or dimension tables. The most common change data involves insertions of new facts. Insertions of new dimensions and modifications of dimensions are less common but are still important to capture.

Depending on the characteristics of the source system, change data can be classified as cooperative, logged, queryable, or snapshot as summarized in Table 2-1. Cooperative change data involves notification from the source system about changes. The notification typically occurs at transaction completion time using a trigger. Cooperative change data can be input immediately into a data warehouse or placed in a queue or staging area for later processing possibly with other changes. Because cooperative change data involves modifications to both a source system and the data warehouse, it has traditionally been the least common format for change data. However, as data warehouse projects mature and legacy systems are redeveloped, cooperative change data is becoming more common.

Logged change data involves files that record changes or other user activity. For example, a transaction log contains every change made by a transaction, and a Web log contains page access histories (called clickstreams) by website visitors. Logged change data usually involves no changes to a source system as logs are readily available for most source systems. However, logs can contain large amounts of irrelevant data. In addition, derivation of the relevant data needed for the data warehouse can require matching related log records, a resource-intensive task. Logged change data is most commonly used for the customer relationship subset of a data warehouse.

As its name implies, queryable change data comes directly from a data source via a query. Queryable change data requires time stamping in the data source. Since few data sources contain timestamps for all data, queryable change

data usually is augmented with other kinds of change data. Queryable change data is most applicable for fact tables using columns such as order date, shipment date, and hire date that are stored in operational data sources.

Snapshot change data involves periodic dumps of source data. To derive change data, a difference operation uses the two most recent snapshots. The result of a difference operation is called a delta. Snapshots have historically been the most common form of change data because of applicability. Snapshots are the only form of change data without requirements on a source system. Because computing a snapshot can be resource intensive, there may be constraints about the time and frequency of retrieving a snapshot.

Table 2-1: Summary of the Change Data Classification

Change Type	Description	Evaluation
Cooperative	Source system notification using triggers	Requires modifications to source systems
Logged	Source system activity captured in logs	Readily available but significant processing to extract useful data
Queryable	Source system queries using timestamps	Requires timestamps in the source data and non-legacy source systems
Snapshot	Periodic dumps of source data augmented with difference operations	Resource intensive to create and significant processing for difference operations; no source system requirements so useful for legacy data

2.1.2 Workflow for Maintaining a Data Warehouse

Maintaining a data warehouse involves a variety of tasks that manipulate change data from source systems. Figure 2.1 presents a generic workflow that organizes the tasks. The preparation phase manipulates change data from individual source systems. Extraction involves the retrieval of data from an individual source system. Transportation involves movement of the extracted data to a staging area. Cleaning involves a variety of tasks to standardize and improve the quality of the extracted data. Auditing involves recording results of the cleaning process, performing completeness and reasonableness checks, and handling exceptions.

The integration phase merges the separate, cleaned sources into one source. Merging can involve the removal of inconsistencies among the source data. Audit processes record results of the merging process, performs completeness and reasonableness checks, and handles exceptions.

The update phase involves propagating the integrated change data to various parts of the data warehouse including fact and dimension tables, materialized views, stored data cubes, and data marts. After propagation, notification can be sent to user groups and administrators.

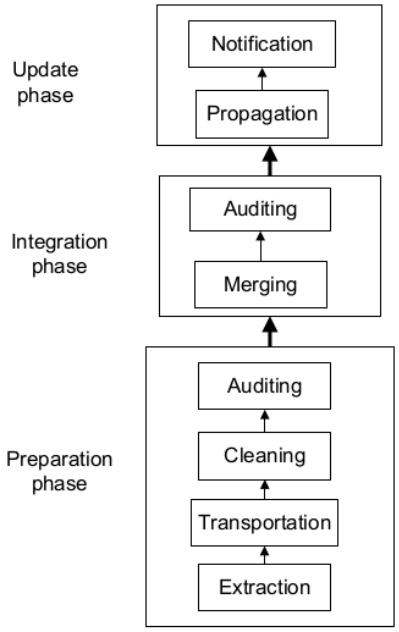


Figure 2.1: Generic Workflow for Data Warehouse Maintenance

The preparation and integration phases should resolve data quality problems as summarized in Table 2-2. Data from legacy systems are typically dirty, meaning that they may not conform to each other or to enterprise-wide data quality standards. If directly loaded, dirty data may lead to poor decision making. To resolve data quality problems, the auditing task should include exception handling. Exceptions can be noted in a log file and then manually addressed. Over time, exceptions should decrease as data quality standards are improved on internal data sources.

Table 2-2: Typical Data Quality Problems

Multiple identifiers: some data sources may use different primary keys for the same entity such as different customer numbers
Multiple names: the same field may be represented using different field names
Different units: measures and dimensions may have different units and granularities.
Missing values: data may not exist in some databases; to compensate for missing values, different default values may be used across data sources
Orphaned transactions: some transactions may be missing important parts such as an order without a customer
Multipurpose fields: some databases may combine data into one field such as different components of an address
Conflicting data: some data sources may have conflicting data such as different customer addresses
Different update times: some data sources may perform updates at different intervals

In addition to exception handling, the auditing task should include completeness checks and reasonableness checks. A completeness check counts the number of reporting units to ensure that all have reported during a given period. A reasonableness check determines whether key facts fall in predetermined bounds and are a realistic extrapolation of previous history. Exceptions may require reconciliation by business analysts before propagation to a data warehouse.

The generic workflow in Figure 2.1 applies to both the initial loading and the periodic refreshment of a data warehouse. The initial loading often requires a long period of data cleaning to resolve data quality problems. A goal of initial loading is to discover data quality problems and resolve them. The refresh process typically varies among data sources. The workflow for the refresh process should be customized to fit the requirements of each data source. For example, auditing may be minimized for high-quality data sources. Data integration tools support graphical workflow

modeling to allow organizations to customize workflows. Section 2.1.4 provides more details about workflow modeling as well as other features of data integration tools.

2.1.3 Data Cleaning Techniques

Data cleaning is an important part of the data integration workflow to resolve the data quality problems listed in Table 2-2. This section explains common data cleaning techniques and demonstrates their usage to resolve data quality problems. Data integration tools, presented in the next section, support nonprocedural specification for many data cleaning techniques. This section covers parsing of multipurpose fields, correcting and standardizing values for missing and inconsistent values, and entity matching for multiple identifiers.

String Parsing with Regular Expressions

Parsing decomposes complex objects into their constituent parts. For data integration, parsing is important for decomposing multipurpose text data into individual fields. Parsing of physical addresses, phone numbers, and email addresses are typical transformations for marketing data warehouses. To facilitate target marketing analysis, these composite fields should be decomposed into standard parts. For example, data sources that contain addresses in a single field typically require parsing into standard fields such as the street number, street, city, state, country, and postal code. Figure 2.2 demonstrates parsing of a customer's name and address into constituent fields.

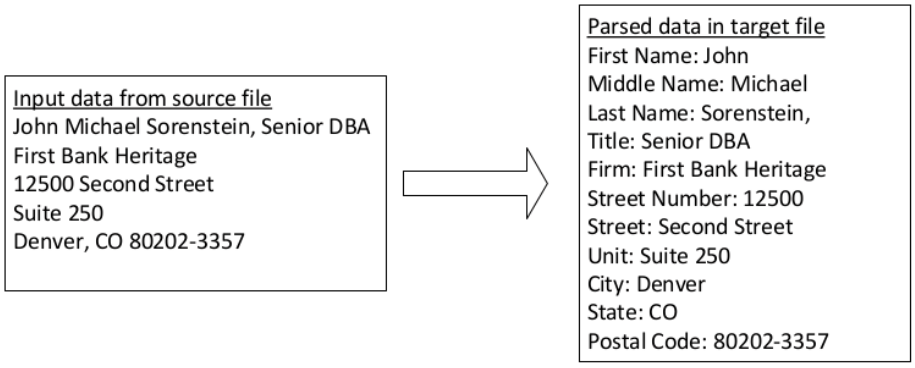


Figure 2.2 Parsing Name and Address into Constituent Fields

Regular expressions are pattern specifications that are important elements of parsing in data integration tasks. Chapter 4 provided a brief introduction to inexact matching in SQL SELECT statements although this subsection provides more details because of the importance of regular expressions in data integration. A regular expression (or regex for short) contains literals, metacharacters, and escape characters together that define a search pattern. A literal is a character to match exactly. Table 2-3 contains simple regular expressions using only literals. Note that the literals match the first appearance of the characters in the search string. The quotation marks are not part of the regular expressions or search strings. A regular expression engine compares a regular expression to a pattern, providing a true/false response along with an optional indication of the first matching character.

Table 2-3: Regular Expression Examples with Literals Only

Regular Expression	Search String	Evaluation
"a"	"John Michael"	True; matches the 9th position
"a"	"Senior DBA"	False; match is case sensitive
"Co"	"Denver, CO"	False; "o" not matched
"Suite 2"	"Suite 250"	True; matches 1st position; spaces are literals

Metacharacters, characters with special meaning within a search pattern, provide the power of regular expressions. Table 2-4 provides a brief explanation of prominent metacharacters. The iteration or quantifier metacharacters, ?, *, +, and {}, support matches on consecutive characters. The range metacharacters, [], match a single character from a range of specified characters. The position metacharacters or anchors, ^, $, and . (period character), support match-

ing in specified places in a string. The alteration metacharacter, |, supports optional parts of search patterns. Table 2-5 provides examples for each metacharacter along with comments about the examples.

Table 2-4: Meaning of Important Metacharacters

Metacharacter	Type	Meaning
?	Iteration	Matches preceding character 0 or 1 time
*	Iteration	Matches preceding character 0 or more times
+	Iteration	Matches preceding character 1 or more times
{n}	Iteration	Matches preceding character exactly n times
{n,m}	Iteration	Matches preceding character at least n times and at most m times
[]	Range	Matches enclosed character one time
^	Position	Matches the following search string at the beginning of the search string; using ^ only has meaning as the first character in a regular expression; ^ used inside [] has a different meaning
^	Range	Negation of search pattern if ^ is inside []
$	Position	Matches the preceding search pattern at the end of the search string; $ only has meaning as the last character in a regular expression.
.	Position	Matches any character except a newline character at the specified position only
|	Alteration	Matches either pattern to the left or right of the | character.

The backslash (\) escape character removes the meaning of a metacharacter allowing a metacharacter to be used as a literal. For example, the regular expression “abc*” matches the search string “ddd abc*” but not “fabc”. The backslash only has the escape meaning before a metacharacter. Otherwise, the backslash followed by a literal can refer to predefined set of characters known as a shorthand character class. For example “\d” refers to the digits 0 to 9 and “\w” refers to word characters (letters, digits, and underscore).

Table 2-5: Regular Expression Examples with Metacharacters

Regular Expression	Search Strings	Evaluation
“colou?r”	“color”, “colour”	Matches both search strings; Matches preceding character 0 times in first search string
“tre*”	“tree”, “tread”, “trough”	Matches all three search strings; Matches preceding character 0 times in third search string
“tre+”	“tree”, “tread”, “trough”	Does not match the third search string; The + metacharacter requires matching on at least one character.
“[abcd]”	“dog”, “fond” , “pen”	Matches first two strings but not the third string; Matches strings that contain any enclosed character
“[0-9]{3}-[0-9]{4}”	“123-4567”, “1234-567”	Matches first string but not the second string; Regular expression uses iteration and range metacharacters.
“ba{2,3}b”	“baab”, “baaab”, “bab”, “baaaab”	Matches first two strings but not the last two strings; four consecutive “a” characters do not match in the last search string.
“^win”	“erwin”, “window”	Does not match the first search string because “win” does not appear in the beginning of the search string
“win$”	“erwin”, “window”	Does not match the second search string because “win” does not appear at the end of the search string

"[^0-9]+"	"123", "abc", "a456"	Matches the second and third search strings; ^ negates the string pattern inside the [] matching any non-digit.
"abc.e*"	"fabc", "fabcd", "fabcee"	Matches the second and third search strings; Regular expression requires a character following "abc".
"dog|cat|frog"	"a dog", "cat friend", "frogman"	Matches all three search strings because of the alteration metacharacters in the pattern

Regular Expression: a pattern specification that is important for parsing of multipurpose text fields in data integration tasks. A regular expression (or regex for short) contains literals (exactly matching characters), metacharacters (special meaning characrters), and escape characters (remove special meaning of metacharacters).

For complex matching requirements, it is useful to match different subparts or groups in a search string. Groups facilitate parsing because parts of a string can be matched and used for later processing. For example, a date string can be parsed into year, month, and day components using groups. The parentheses metacharacters () provide matching on groups in a search string. Matching with groups provides a list of matched groups. For example, the regular expression "a(b*)c" matched against the search string "abbc" produces two matched groups: (Group 0) "abbc" matching the entire regular expression and (Group 1) "bb" matching the regular expression group "(b*)". As additional features, a pattern can have more than one group and groups can be nested. For example, the regular expression "(d(e*))*(c+)" contains group 1 with the pattern "(d(e*))", group 2 with the pattern "(e*)", and group 3 with the pattern "(c+)". In addition, the entire pattern is group 0.

This introduction only scratches the surface of the power and complexity of regular expressions. There are other important aspects such as conditional matching and additional grouping features. Data integration tools typically support regular expressions along with a replacement function to substitute specified contents for matching parts of a search string. Libraries of regular expressions support parsing of common fields such as names, addresses, phone numbers, and email addresses.

Correcting and Standardizing Values

Correcting values involves resolution of missing and conflicting values. Reasonable resolution requires understanding the reason for missing values. Missing values in incomplete drill-downs, incomplete roll-ups, and incomplete dimension-fact relationships can usually be resolved through default values as explained in Chapter 1. Missing values in these cases are usually due to unallocated relationships. For example, a department is not allocated to a division, a division lacks departments, or a sale involves an anonymous customer. In all these situations, default values for the unattached entities are appropriate. Missing values inapplicable to an entity can often be resolved through default values. For example, unmarried students and customers will not have values for attributes related to spouses. The default value could be the same value as the corresponding value for the entity. For example, student age and spouse age would be set to the same value for unmarried students.

Missing values that are unknown rather than inapplicable are more difficult to resolve. For example, missing dates of birth, addresses (or parts of an address), and grade point averages are more difficult to resolve. One approach to unknown values involves typical values. For unknown numeric values, a median or average value can be used. For unknown non numeric values, the mode (most frequent value) can be used. Typical values can be refined through conditional probability calculations. The typical value would the one most likely given other attribute values of the entity. More complex approaches will predict missing values using data mining algorithms. These approaches extend the conditional probability approach by additional decision criteria using a large sample.

Detailed investigations possibly conducted using search services can resolve some cases of unknown values and conflicting values. Figure 2.3 demonstrates the result of an investigation to determine the middle name and street address in an employee record. A map and knowledge about the location of a building was used to obtain the street address. Other employee files were used for the missing middle name. For conflicting values, the more recent value is preferred although a data source may lack timestamps to determine the update time. Without a timestamp, the more credible source may be preferred.

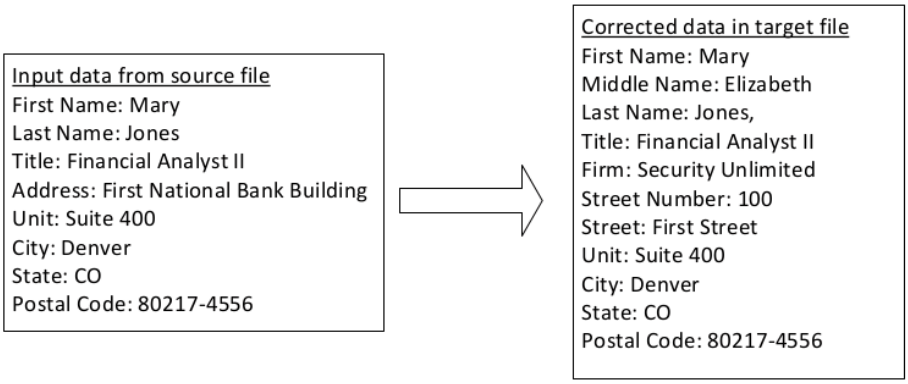


Figure 2.3 Completed Missing Values through an Investigation

Standardization involves business rules to transform values into preferred representations. Both standard and custom business rules can be developed. Standardization is typically applied to units of measure and abbreviations. To augment searching, business rules can provide alternative values in the results of data warehouse queries. Data standardization services can be purchased for names, addresses, and product details although customization may be necessary.

The approaches described in this subsection are reactive, attempting to resolve problems occurring in existing data sources. Proactive approaches can be more cost effective if changes in data collection procedures can be made in different parts of an organization. Standards can be facilitated by XML schemas with rules about data interchange. It may even be possible to apply standards to external data sources if external users derive benefits from a data warehouse.

Entity Matching

Entity matching refers to identification of duplicate records when no common, reliable identifier exists. The classic application involves identification of duplicate customers in lists from different firms. Because a common identifier does not exist, duplicates must be identified from other common attributes such as names, address components, phone numbers, and ages. Because these common attributes come from different data sources, inconsistency and non standard representations may exist complicating the matching process.

Despite the difficulty of the entity matching problem, it is important in many applications. Marketing is the most prominent area as firms are often interested in expanding their customer bases. Merging of firms typically triggers a major customer matching effort. Law enforcement agencies need to link crimes to suspects and combine aliases into one suspect. Fraud detection must resolve individuals who claim benefits under different identifiers when the individuals are the same person. For example, the same person may fraudulently file multiple tax returns to receive tax credits. Other applications involve insurance industry needs to link crash and injury reports and gene identification in lab and research reports.

To obtain a more precise understanding of the problem, the outcomes of comparing two cases should be understood. The outcomes of entity identification are similar to standard classification problems as shown in Table 2-6. The rows in the Table 2-6 represent predictions and the columns represent actual results of matching two records for duplication. A true match involves a predicted match and an actual match allowing the two records to be combined correctly. A false match involves a predicted match but an actual non match resulting in two records combined that should have remained separate. A false non match involves a prediction of non match but an actual match resulting in two records remaining separate that should be combined. A true non match involves a prediction of non match and actual non match resulting in two separate records remaining separate. The “possible match” situations involve predictions with too much uncertainty. Investigation is required to resolve cases with high uncertainty in match prediction.

Table 2-6: Entity Matching Outcomes

Predicted \ Actual	*Match*	*Non Match*
Match	True match	False match
Possible Match	Investigation	Investigation
Non Match	False non match	True non match

Entity Matching: identification of duplicate records when no common, reliable identifier exists. Because a common identifier does not exist, duplicates must be identified from other common attributes such as names, address components, phone numbers, and ages. The result of matching two records may be a true match, false match, false non match, true non match, or additional investigation.

In an economic sense, entity matching procedures should balance the benefits of consolidated entity lists (true matches and true non matches) against the costs of incorrect actions (false matches and false non matches) plus investigation costs. The costs of false matches are typically the largest as a false match eliminates a potential customer. Determining uncertainty levels leading to investigations is also important if investigation cost is substantial. Investigation costs may be labor intensive so the costs are likely more than the cost of false non matches.

Many approaches to entity identification have been developed but no dominant approach has emerged. Data mining tools typically include methods for entity identification. In addition, commercial services with customization to individual data source requirements can match entities but usually with relatively high cost. Before deciding on a solution, a firm should investigate data quality levels in the data sources to consolidate. To improve entity identification results, investment to improve consistency and incompleteness in underlying data sources is usually worthwhile.

Distance measures for string comparisons are important components of entity matching solutions. Comparing text for similarity is necessary in most applications of entity matching. Often, text fields will not match exactly so similarity assessment must be performed. Three common measures of string similarity are edit distance, N-gram distance, and phonetic distance. Edit distance is the number of deletions, insertions, or substitutions required to transform a source string into a target string. *N*-gram distance breaks text into subsequences of length *n* and then measures the intersections among the *n*-grams. Phonetic distance codes words into standard consonant sounds. Phonetic distance has many applications in law enforcement to account for different name spellings. Data integration tools typically include distance measures that can be used for entity matching.

Customer data integration is a generalization of entity matching for marketing data warehouses. Customer data integration promotes the sharing of customer data between both front-office and back-office processes as well as with the business-to-consumer and business-to-business web-enabled systems. Customer data integration provides an enterprise customer hub that serves as a central repository of customer information reconciled from multiple data sources, both internal and external. The customer data integration market is comprised of process and technology solutions for recognizing a customer at any business point while aggregating and delivering timely and accurate data about customers.

2.1.4 Data Integration Architectures and Tools

To support the complexity of data warehouse maintenance and loading, software products for data integration have been developed. Data integration software have evolved from disparate collections of tools to reduce tedious coding to integrated development environments supporting a full range of data integration tasks. Table 2-7 lists typical features of enterprise data integration tools. Most data integration tools use rule and graphical specifications rather than procedural coding to indicate logic and actions. Some tools can generate code that can be customized for more flexibility. Graphical specification of complex maintenance workflows support design and monitoring of maintenance jobs. Change data capture typically uses a “publish and subscribe” model to control change data availability and notify subscribers about change data availability.

Data Integration Tools: software tools for extraction, transformation, and loading of change data from data sources to a data warehouse. Enterprise tools provide integrated development environments supporting a full range of data integration tasks.

Table 2-7: Description of Typical Features of Enterprise Data Integration Tools

Feature	Description
Workflow specification	Graphical specification of workflow designs and executions
Transformation specification	Non procedural specification of data transformations for data types, calculations, lookups, aggregations, and string manipulations
Job management	Graphical display of workflow progress; Tools for optimized scheduling of jobs using parallel processing
Data profiling	Non procedural specification to understand data characteristics and data quality levels
Change data capture	Synchronous (via triggers) and asynchronous capture of source data
Integrated development environment	Graphical development environment for all data integration functions along with team support and version control
Repository	Database for all data integration specifications, jobs, users, and data capture results
Database/file connectivity	Connection to a large collection of databases and file types

The vibrant marketplace for data integration products and services contains third party vendors and DBMS vendors offering both proprietary and open source products along with a wide range of services. Third party vendors emphasize support for a variety of DBMS products. DBMS vendors leverage relational database support for data warehouse implementation. Open source products are typically base products accompanied by subscription services for enterprise products. At the end of 2012, a Gartner report estimated the data integration market size at $4 billion in 2013 with annual growth rate of 10.3 percent to grow to $6 billion in 2017. DBMS vendors (Oracle, IBM, and Microsoft) have strong market penetration along with third party vendors Informatica and SAP Business Objects. Beyond these market leaders, the market contains 10 to 20 additional firms with reasonable market penetrations. The overall marketplace is fluid with acquisitions and new product developments likely having strong influence on future market structure.

Two major architectures dominate enterprise data integration tools. The Extraction, Transformation, and Loading (ETL) architecture performs transformation before loading as shown in Figure 2.4a. Transformations and data quality checks are performed by a dedicated ETL engine before loading transformed data into target data warehouse tables. The Extraction, Loading, and Transformation (ELT) architecture performs data transformations and data quality checks after loading as depicted in Figure 2.4b. The ELT architecture relies on a relational DBMS to generate SQL statements and procedures and move data between tables. ETL architecture supporters emphasize DBMS independence of ETL engines, while ELT architecture supporters emphasize superior optimization technology in relational DBMS engines. ETL architectures can usually support more complex operations in single transformations than ELT architectures, but ELT architecture may use less network bandwidth. Some data integration tools support both architectures so the distinction between the architectures may blur somewhat in the future. In addition, a combination of ETL and ELT processing may provide better performance for enterprise warehouses so the demand for both architectures should grow without either architecture dominating.

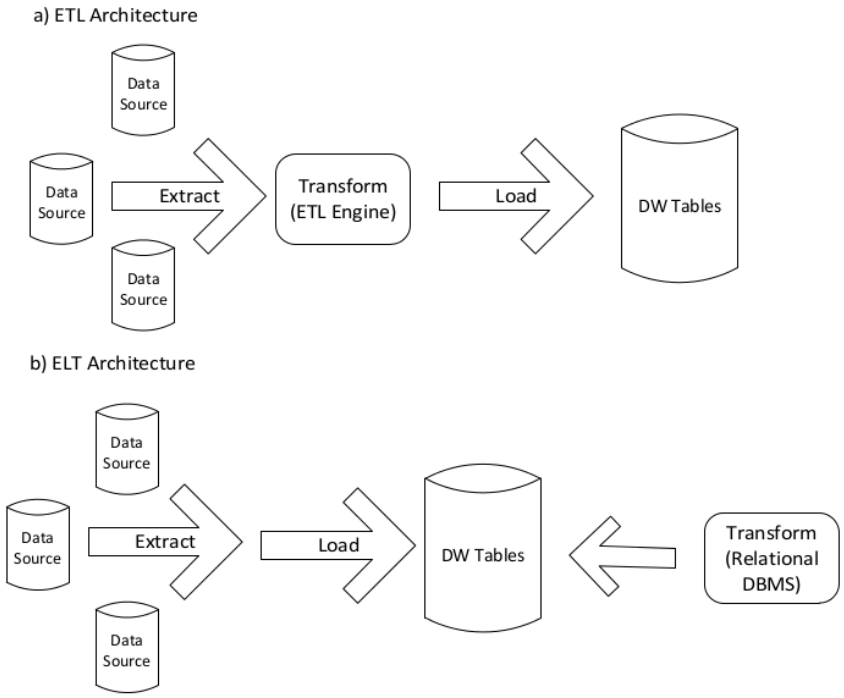


Figure 2.4 ETL versus ELT Architectures for Data Integration

Talend Open Studio

To provide more details about data integration products, the features of Talend Open Studio are presented along with an example of its usage. Talend Open Studio for Data Integration, an open source product offered by Talend, is a base product designed for modest size data integration needs. Talend, although not a leader in the data integration market, is an innovative secondary firm. To support enterprise data warehouses, Talend offers subscription products and services including Talend Open Studio for Big Data with parallel and real-time processing options, Talend Open Studio for Data Quality, and Talend Open Studio for Master Data Management for enterprise data standards. The Talend Open Studio platform provides common features for job design, repository of meta data, deployment, execution, and monitoring.

Talend Open Studio for Data Integration supports graphical business modeling and job design, a palette of data transformation components, a metadata repository, job execution, and database connectivity. The business modeling component uses standard flowchart symbols to document business processes involved in data integration efforts. The job design component connects transformation components to implement complex data integration tasks. Table 2-8 lists important categories of Talend components available in a job design. Standard component details are specified in property windows. More advanced development can be done in Java procedures to extend component capabilities. The job execution component makes connections to data sources (databases, files, and XML data), executes data transformations, and displays the results in graphical notation. The repository component allows retrievals of job designs, business models, components, and job executions.

Table 2-8: Description of Component Categories in Talend Open Studio

Component Category	Description
Data quality	String matching, distance comparisons, lookup match, uniqueness comparison
Database	Reading and writing to databases
File	Reading and writing to files
Processing	Sorting, denormalization, aggregation, regular expression extraction, filtering, type conversions
ELT	Transformations in a target database such as aggregate calculations, SQL statement execution, and row filtering
Internet	Transformations involving standard internet protocols for email, web pages, file transfers, sockets, and news streams
XML	Schema validation, formatting, parsing

To clarify job design features and components, an example job design is presented. A Talend job design involves a number of components connected to process work. In Figure 2.5, the Excel data source (SSExcelSource) contains rows to be loaded into the *SSSales* table, the fact table of the Store Sales data warehouse. The first component in the flow is the Excel file, a tFileInputExcel component. The file input is processed by the tSchemaComplianceCheck component, a data quality component. Null value and data type checks are performed in the tSchemaComplianceCheck component. The output of the tSchemaComplianceCheck component is further processed by the tMap component. The tMap component performs joins on four dimension tables (*SSCustomer*, *SSStore*, *SSTimeDim*, and *SSItem*) to ensure valid foreign keys. The tMap component uses four tOracleInput components to perform the joins. The output of the tMap processing is loaded into the *SSSales* table, a tOracleOutput component. A commercial job flow would also have outputs for rejected records. The job in Figure 2.5 only has outputs for accepted records. The tSchemaComplianceCheck and tMap components would both be connected to an additional output for rejected records.

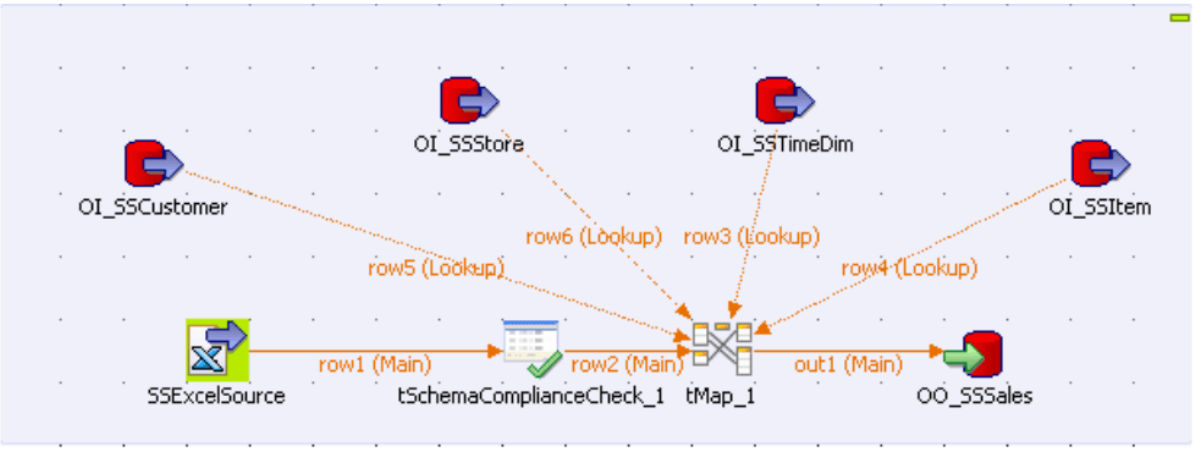


Figure 2.5 Example Job Design using Talend Open Studio

Component details are specified in a non-procedural manner using property windows and graphical displays. For example, the tMap component uses a graphical display to support join specification. In Figure 2.6, the fields in the Excel data source are mapped to columns in the *SSTimeDim* Oracle table. The columns in the top part (row1) of Figure 2.6 are from the input file. The columns in the bottom part (row2) of the window are the *SSTimeDim* table. A "drag and drop" method is used to match the columns in the data source and table.

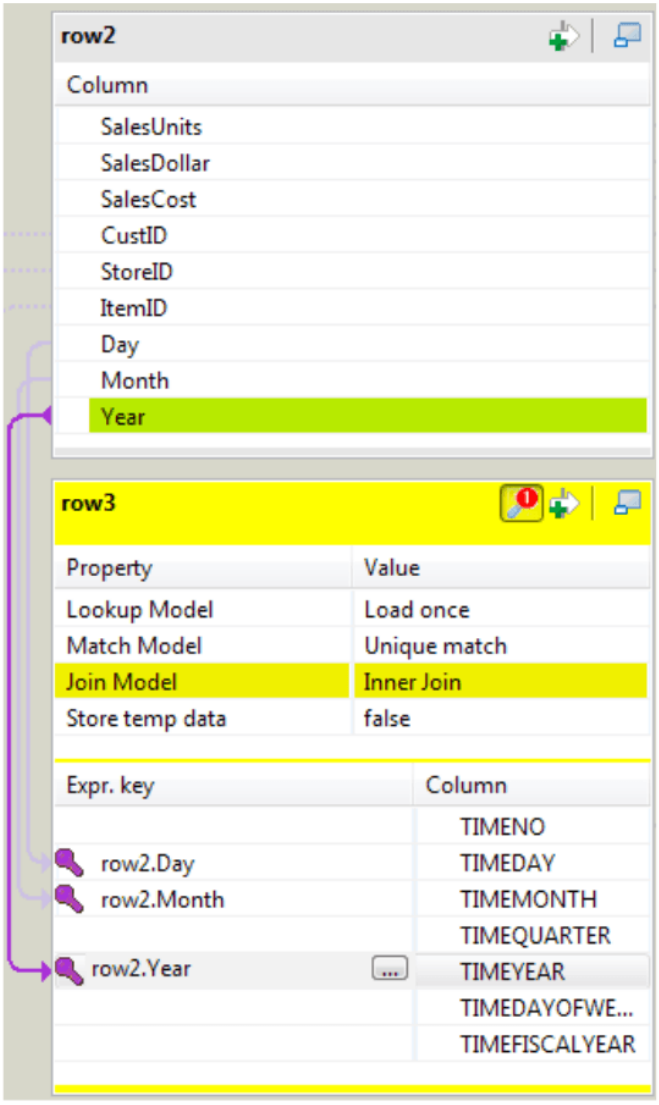


Figure 2.6 Talend Join Specification for Time Columns in the tMap Component

Talend Open Studio provides graphical display of job executions as depicted in Figure 2.7. The screen snapshot was taken from the job design pane after executing the job. The job execution display shows the number of rows processed along with processing times. Figure 2.7 shows that the Excel input file contained 12 rows. The tSchemaCompliance-Check component rejected two rows for null value or data type violations, passing 10 rows to the tMap component. The tMap component rejected two rows for foreign key violations, passing 8 rows to the tOracleOutput component for loading into the *SSSales* fact table.

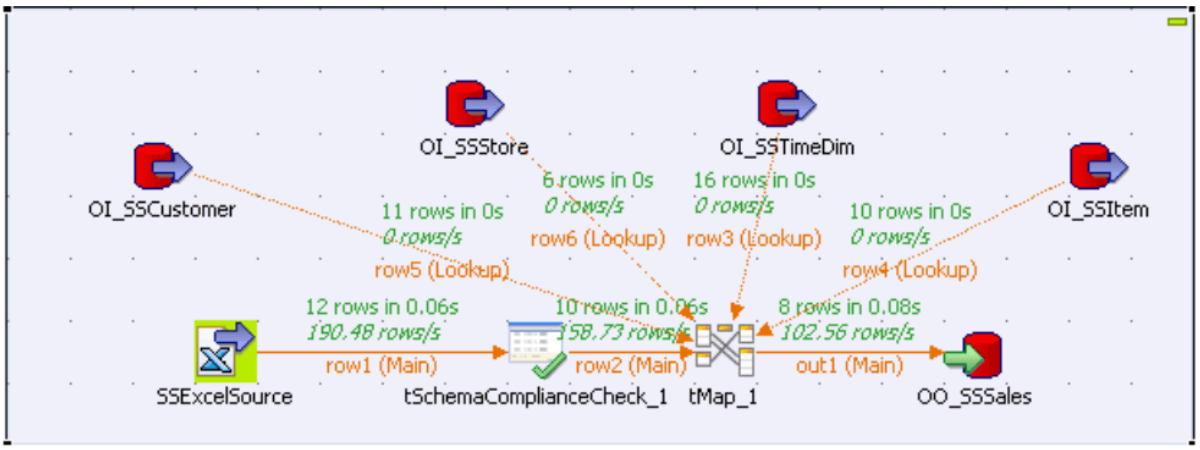


Figure 2.7 Example Job Execution using Talend Open Studio

Oracle Data Integration Tools

Oracle offers two major data integration tools along with a collection of independent tools. Unlike Talend, Oracle is one of the market leaders in the data integration market. The Oracle Data Integrator is a recent tool developed as a result of Oracle's acquisition of BEA Systems in 2007. The Oracle Warehouse Builder was first released in 2000 with the current version as 11gR2. Oracle intends to merge both products in the coming years although there is some confusion about the appropriateness of each tool for different data integration environments before the product merger completes. The Oracle Data Integrator provides better support for heterogeneous environments along with declarative knowledge modules for transformations. The Oracle Warehouse Builder provides closer integration with Oracle databases, applications, and business intelligence tools. Both tools are complex with too many details to present here.

In addition to enterprise data integration tools, Oracle provides many independent tools that may are appropriate to smaller organizations without a need for an integrated, enterprise tool. In addition, some of the features of the independent tools have been incorporated into the enterprise products. The following list summarizes some independent integration tools and SQL extensions available in Oracle.

- SQL*Loader is a long standing Oracle utility for loading data from text files into Oracle tables. SQL*Loader provides data type conversion, conditional loading of data, simple NULL value handling, transformation of data before loading using SQL functions, and direct-path loading for improved performance.
- The proprietary Oracle SQL MERGE statement provides the ability to update or insert a row conditionally into a table.
- The proprietary Oracle SQL multiple table INSERT statement provides the ability to segregate data on logical attributes for insertion into different target tables.
- Oracle Change Data Capture supports synchronous (via triggers) or asynchronous capture (via log files) of change data. The Change Data Capture feature uses a Publish and Subscribe model[1] to control change data availability and notify subscribers about change data availability. Oracle Change Data Capture has been incorporated into the Oracle Data Integrator.
- The Oracle Data Pump enables very high-speed movement of data and metadata from one database to another using import and export utilities.

2.1.5 Managing the Refresh Process

Refreshing a data warehouse is a complex process that involves the management of time differences between the updating of data sources and the updating of the related data warehouse objects (tables, materialized views, data cubes, data marts). In Figure 2.8, valid time lag is the difference between the occurrence of an event in the real world (valid time) and the storage of the event in an operational database (transaction time). Load time lag is the difference between

1 Publish and Subscribe model has been discontinued in Oracle 12 Database.

transaction time and the storage of the event in a data warehouse (load time). For internal data sources, there may be some control over valid time lag. For external data sources, there is usually no control over valid time lag. Thus, a data warehouse administrator has most control over load time lag.

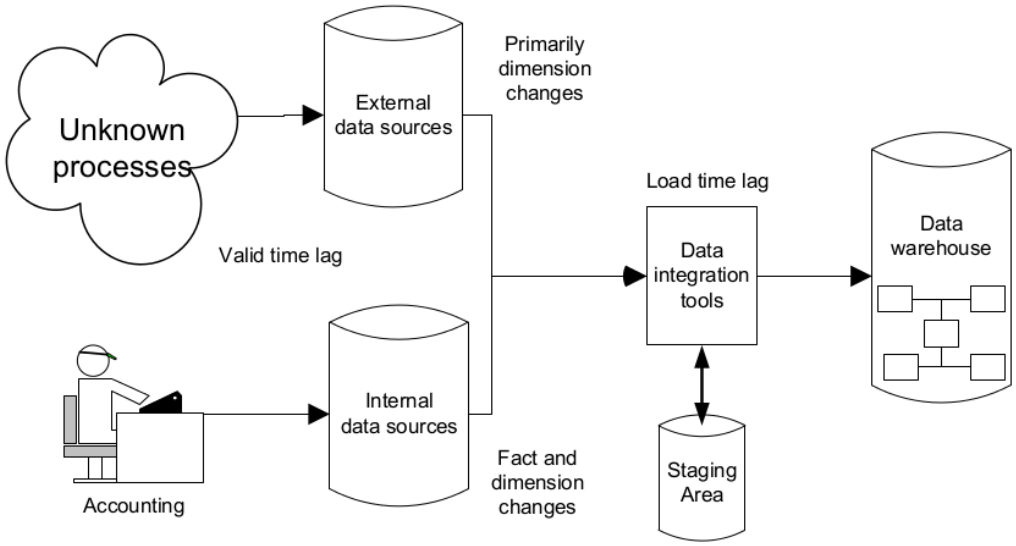


Figure 2.8: Overview of the Data Warehouse Refresh Process

Figure 2.8 implies that data sources can change independently leading to different change rates for fact and dimension tables. Fact tables generally record completed events such as orders, shipments, and purchases with links to related dimensions. For example, inserting a row in a sales fact table requires foreign keys that reference dimension tables for customers, stores, time, and items. However, updates and insertions to the related dimension tables may occur at different times than the fact events. For example, a customer may move or an item may change in price at different times than orders, shipments, or inventory purchases. As a result of the different change rates, a data warehouse administrator should manage the load time lag separately for dimension tables and fact tables.

The primary objective in managing the refresh process is to determine the refresh frequency for each data source. The optimal refresh frequency maximizes the net refresh benefit defined as the value of data timeliness minus the cost of refresh. The value of data timeliness depends on the sensitivity of decision making to the currency of the data. Some decisions are very time sensitive such as inventory decisions. Organizations along a supply chain attempt to minimize inventory carrying costs by stocking goods as close as possible to the time needed. Other decisions are not so time sensitive. For example, the decision to close a poor performing store would typically be done using data over a long period of time.

Fisher and Berndt (2001) proposed a method to measure the value of data timeliness. They defined a measure of the error rate of a data warehouse with respect to a given workload of queries. To determine the error rate, the data volatility of queries and dimensions is estimated. Even if a data warehouse stores individual-level data, most queries will involve aggregations, not retrieval of the individual-level data. For example, most queries on the store sales schema from Chapter 16 would involve the item brand or category, not the individual items. Given an assessment of the error rate, the value of data timeliness can be measured using the frequency or other weighting for data warehouse queries.

The cost to refresh a data warehouse includes both computer and human resources. Computer resources are necessary for all tasks in the maintenance workflow. Human resources may be necessary in the auditing tasks during the preparation and integration phases. The level of data quality in source data also affects the level of human resources required. The development effort to use data integration tools and write custom software is not part of the refresh cost unless there is ongoing development cost with each refresh. An important distinction involves the fixed cost and the variable cost of refreshment. Large fixed cost encourages less frequent refresh because the fixed cost occurs with each refresh. Fixed cost may include startup and shutdown effort as well as resource rental.

Along with balancing the value of timeliness against the cost of refresh, the data warehouse administrator must satisfy constraints on the refresh process as summarized in Table 2-9. Constraints on either the data warehouse or a source system may restrict frequent refresh. Source access constraints can be due to legacy technology with restricted scalability for internal data sources or coordination problems for external data sources. Integration constraints often involve identification of common entities such as customers and transactions across data sources. Completeness/consistency constraints can involve maintenance of the same time period in change data or inclusion of change data from each data source for completeness. Data warehouse availability often involves conflicts between online availability and warehouse loading.

Table 2-9: Summary of Refresh Constraints

Constraint Type	Description
Source access	Restrictions on the time and frequency of extracting change data
Integration	Restrictions that require concurrent reconciliation of change data
Completeness/consistency	Restrictions that require loading of change data in the same refresh period
Availability	Load scheduling restrictions due to resource issues including storage capacity, online availability, and server usage

2.2 Extensions to SQL for Multidimensional Data

Beginning in SQL:1999 and continuing through SQL:2011, new summarization capabilities have been available in the GROUP BY clause. These features are an attempt to unify the proliferation of proprietary data cube extensions, although they do not obviate the need for visual tools that directly support data cube operations. The extensions involve the ability to produce summary totals (CUBE and ROLLUP operators) as well as more precise specification of the grouping columns (GROUPING SETS operator). This section describes these new parts of the GROUP BY clause using Oracle as an example DBMS that implements the standard SQL features.

The examples in the remainder of this chapter use the Store Sales Data Warehouse presented in Chapter 16. The ERD is shown in Figure 2.9 for ease of reference.

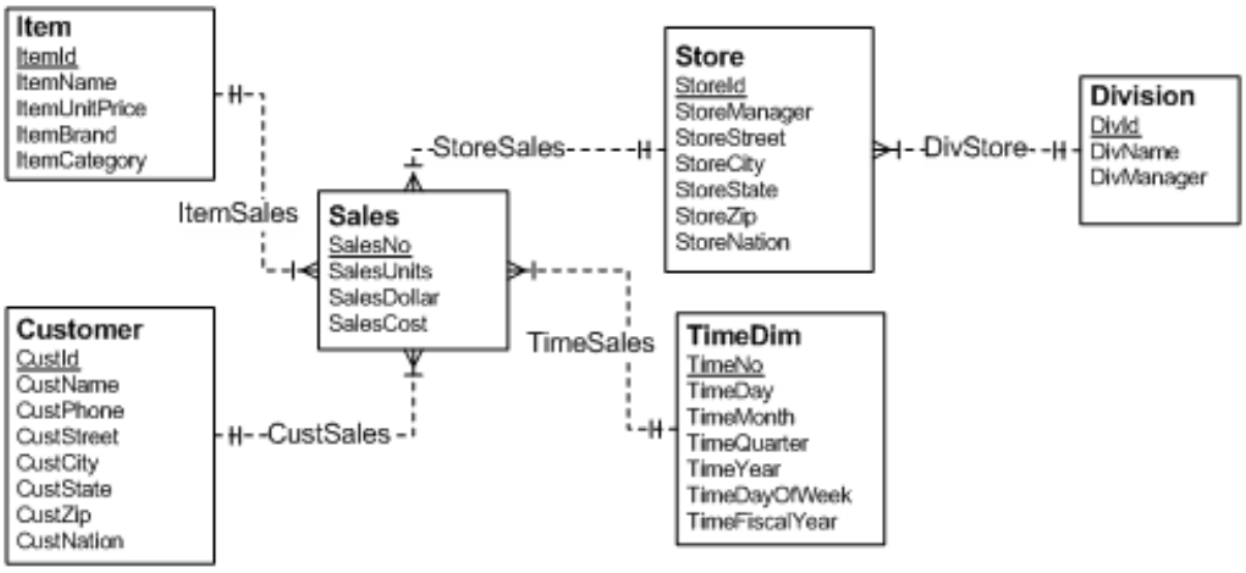


Figure 2.9: ERD Snowflake Schema for the Store Sales Data Warehouse

2.2.1 CUBE Operator

The CUBE operator clause produces all possible subtotal combinations in addition to the normal totals shown in a GROUP BY clause. Because all possible subtotals are generated, the CUBE operator is appropriate to summarize columns from independent dimensions rather than columns representing different levels of the same dimension. For

example, the CUBE operator would be appropriate to generate subtotals for all combinations of month, store state, and item brand. In contrast, a CUBE operation to show all possible subtotals of year, month, and day would have limited interest because of the hierarchy in the time dimension.

CUBE Operator: an operator that augments the normal GROUP BY result with all combinations of subtotals. The CUBE operator is appropriate to summarize columns from independent dimensions rather than columns representing different levels of a single dimension.

To depict the CUBE operator, Example 2.1 displays a SELECT statement with a GROUP BY clause containing just two columns. Only six rows are shown in the result so that the effect of the CUBE operator can be understood easily. With two values in the *StoreZip* column and three values in the *TimeMonth* column, the number of subtotal combinations is six (two *StoreZip* subtotals, three *TimeMonth* subtotals, and one grand total) as shown in Example 2.2. Blank values in the result represent a summary over all possible values of the column. For example, the row <80111, -, 33000> represents the total sales in the zip code 80111 over all months (- represents a do not care value for the month).

Example 2.1: GROUP BY Clause and Partial Result without Subtotals

```
SELECT StoreZip, TimeMonth, SUM(SalesDollar) AS SumSales
 FROM Sales, Store, TimeDim
 WHERE Sales.StoreId = Store.StoreId AND
       Sales.TimeNo = TimeDim.TimeNo
   AND (StoreNation = 'USA' OR StoreNation = 'Canada')
   AND TimeYear = 2012
 GROUP BY StoreZip, TimeMonth;
```

StoreZip	TimeMonth	SumSales
80111	1	10000
80111	2	12000
80111	3	11000
80112	1	9000
80112	2	11000
80112	3	15000

Example 2.2 (Oracle): GROUP BY Clause and Result with Subtotals Produced by the CUBE Operator

```
SELECT StoreZip, TimeMonth, SUM(SalesDollar) AS SumSales
 FROM Sales, Store, TimeDim
 WHERE Sales.StoreId = Store.StoreId
   AND Sales.TimeNo = TimeDim.TimeNo
   AND (StoreNation = 'USA' OR StoreNation = 'Canada')
   AND TimeYear = 2012
 GROUP BY CUBE(StoreZip, TimeMonth);
```

StoreZip	TimeMonth	SumSales
80111	1	10000
80111	2	12000
80111	3	11000
80112	1	9000
80112	2	11000
80112	3	15000
80111		33000
80112		35000

	1	19000
	2	23000
	3	26000
		68000

With more than two grouping columns, the CUBE operator becomes more difficult to understand. Examples 2.3 and 2.4 extend Examples 2.1 and 2.2 with an additional grouping column (*TimeYear*). The number of rows in the result increases from 12 rows in the result of Example 2.3 without the CUBE operator to 36 rows in the result of Example 2.4 with the CUBE operator. For three grouping columns with M, N, and P unique values, the maximum number of subtotal rows produced by the CUBE operator is $M + N + P + M*N + M*P + N*P + 1$. Since the number of subtotal rows grows substantially with the number of grouped columns and the unique values per column, the CUBE operator should be used sparingly when there are more than three grouped columns.

Example 2.3: GROUP BY Clause with Three Grouping Columns and the Partial Result without Subtotals

```
SELECT StoreZip, TimeYear, TimeMonth, SUM(SalesDollar) AS SumSales
 FROM Sales, Store, TimeDim
 WHERE Sales.StoreId = Store.StoreId
   AND Sales.TimeNo = TimeDim.TimeNo
   AND (StoreNation = 'USA' OR StoreNation = 'Canada')
   AND TimeYear BETWEEN 2012 AND 2013
 GROUP BY StoreZip, TimeYear, TimeMonth;
```

StoreZip	TimeYear	TimeMonth	SumSales
80111	2012	1	10000
80111	2012	2	12000
80111	2012	3	11000
80112	2012	1	9000
80112	2012	2	11000
80112	2012	3	15000
80111	2013	1	11000
80111	2013	2	13000
80111	2013	3	12000
80112	2013	1	10000
80112	2013	2	12000
80112	2013	3	16000

Example 2.4 (Oracle): GROUP BY Clause with Three Grouping Columns and the Result with Subtotals Produced by the CUBE Operator.

```
SELECT StoreZip, TimeYear, TimeMonth, SUM(SalesDollar) AS SumSales
 FROM Sales, Store, TimeDim
 WHERE Sales.StoreId = Store.StoreId
   AND Sales.TimeNo = TimeDim.TimeNo
   AND (StoreNation = 'USA' OR StoreNation = 'Canada')
   AND TimeYear BETWEEN 2012 AND 2013
 GROUP BY CUBE(StoreZip, TimeYear, TimeMonth);
```

StoreZip	TimeYear	TimeMonth	SumSales
80111	2012	1	10000
80111	2012	2	12000
80111	2012	3	11000
80112	2012	1	9000

80112	2012	2	11000
80112	2012	3	15000
80111	2013	1	11000
80111	2013	2	13000
80111	2013	3	12000
80112	2013	1	10000
80112	2013	2	12000
80112	2013	3	16000
80111		1	21000
80111		2	25000
80111		3	23000
80112		1	19000
80112		2	22000
80112		3	31000
80111	2012		33000
80111	2013		36000
80112	2012		35000
80112	2013		38000
	2012	1	19000
	2012	2	23000
	2012	3	26000
	2013	1	21000
	2013	2	25000
	2013	3	28000
80111			69000
80112			73000
		1	40000
		2	48000
		3	54000
	2012		68000
	2013		74000
			142000

The CUBE operator is not a primitive operator. The result of a CUBE operation can be produced using a number of SELECT statements connected by UNION operations, as shown in Example 2.5. The additional SELECT statements generate subtotals for each combination of grouped columns. With two grouped columns, three additional SELECT statements are needed to generate the subtotals. With N grouped columns, $2^N - 1$ additional SELECT statements are needed. Obviously, the CUBE operator is much easier to write than a large number of additional SELECT statements.

Example 2.5: Rewrite Example 2.2 without Using the CUBE Operator

In each additional SELECT statement, a default value (0 for numeric columns and " for text columns) replaces the column in which totals are not generated.

```
SELECT StoreZip, TimeMonth, SUM(SalesDollar) AS SumSales
 FROM Sales, Store, TimeDim
 WHERE Sales.StoreId = Store.StoreId
   AND Sales.TimeNo = TimeDim.TimeNo
   AND (StoreNation = 'USA' OR StoreNation = 'Canada')
   AND TimeYear = 2012
 GROUP BY StoreZip, TimeMonth
 UNION
```

```
SELECT StoreZip, 0, SUM(SalesDollar) AS SumSales
 FROM Sales, Store, TimeDim
 WHERE Sales.StoreId = Store.StoreId
   AND Sales.TimeNo = TimeDim.TimeNo
   AND (StoreNation = 'USA' OR StoreNation = 'Canada')
   AND TimeYear = 2012
 GROUP BY StoreZip
UNION
SELECT '', TimeMonth, SUM(SalesDollar) AS SumSales
 FROM Sales, Store, TimeDim
 WHERE Sales.StoreId = Store.StoreId
   AND Sales.TimeNo = TimeDim.TimeNo
   AND (StoreNation = 'USA' OR StoreNation = 'Canada')
   AND TimeYear = 2012
 GROUP BY TimeMonth
UNION
SELECT '', 0, SUM(SalesDollar) AS SumSales
 FROM Sales, Store, TimeDim
 WHERE Sales.StoreId = Store.StoreId
   AND Sales.TimeNo = TimeDim.TimeNo
   AND (StoreNation = 'USA' OR StoreNation = 'Canada')
   AND TimeYear = 2012;
```

2.2.2 ROLLUP Operator

The SQL ROLLUP operator provides a similar capability to the roll-up operator for data cubes. The roll-up operator for data cubes produces totals for coarser parts of a dimension hierarchy. The SQL ROLLUP operator produces subtotals for each ordered subset of grouped columns to simulate the effects of the roll-up operator for data cubes. For example, the SQL operation `ROLLUP(TimeYear, TimeQuarter, TimeMonth, TimeDay)` produces subtotals for the column subsets <*TimeYear*, *TimeQuarter*, *TimeMonth*>, <*TimeYear*, *TimeQuarter*>, <*TimeYear*> as well as the grand total. As this example implies, the order of columns in a ROLLUP operation is significant.

ROLLUP Operator: an operator that augments the normal GROUP BY result with a partial set of subtotals. The ROLLUP operator is appropriate to summarize levels from a dimension hierarchy.

As the previous paragraph indicates, the ROLLUP operator produces only a partial set of subtotals for the columns in a GROUP BY clause. Examples 2.6 and 2.7 demonstrate the ROLLUP operator to contrast it with the CUBE operator. Note that Example 2.6 contains three subtotal rows compared to six subtotal rows in Example 2.2 with the CUBE operator. In Example 2.7, subtotals are produced for the values in the column combinations <*StoreZip*, *TimeYear*>, <*StoreZip*>, and the grand total. In Example 2.4 with the CUBE operator, subtotals are also produced for the values in the column combinations <*StoreZip*, *TimeYear*>, <*TimeMonth*, *TimeYear*>, <*TimeMonth*>, and <*TimeYear*>. Thus, the ROLLUP operator produces far fewer subtotal rows compared to the CUBE operator as the number of grouped columns and unique values per column increases.

Example 2.6 (Oracle): GROUP BY Clause and Result with Subtotals produced by the ROLLUP Operator

This example should be compared with Example 2.2 to understand the difference between the CUBE and ROLLUP operators.

```
SELECT StoreZip, TimeMonth, SUM(SalesDollar) AS SumSales
 FROM Sales, Store, TimeDim
 WHERE Sales.StoreId = Store.StoreId
   AND Sales.TimeNo = TimeDim.TimeNo
   AND (StoreNation = 'USA' OR StoreNation = 'Canada')
```

```
  AND TimeYear = 2012
GROUP BY ROLLUP(StoreZip, TimeMonth);
```

StoreZip	TimeMonth	SumSales
80111	1	10000
80111	2	12000
80111	3	11000
80112	1	9000
80112	2	11000
80112	3	15000
80111		33000
80112		35000
		68000

Example 2.7 (Oracle): GROUP BY Clause with Three Grouping Columns and the Result with Subtotals Produced by the ROLLUP Operator

This example should be compared with Example 2.4 to understand the difference between the CUBE and ROLLUP operators.

```
SELECT StoreZip, TimeYear, TimeMonth, SUM(SalesDollar) AS SumSales
 FROM Sales, Store, TimeDim
 WHERE Sales.StoreId = Store.StoreId
   AND Sales.TimeNo = TimeDim.TimeNo
   AND (StoreNation = 'USA' OR StoreNation = 'Canada')
   AND TimeYear BETWEEN 2012 AND 2013
 GROUP BY ROLLUP(StoreZip, TimeYear, TimeMonth);
```

StoreZip	TimeYear	TimeMonth	SumSales
80111	2012	1	10000
80111	2012	2	12000
80111	2012	3	11000
80112	2012	1	9000
80112	2012	2	11000
80112	2012	3	15000
80111	2013	1	11000
80111	2013	2	13000
80111	2013	3	12000
80112	2013	1	10000
80112	2013	2	12000
80112	2013	3	16000
80111	2012		33000
80111	2013		36000
80112	2012		35000
80112	2013		38000
80111			69000
80112			73000
			142000

Like the CUBE operator, the ROLLUP operator is not a primitive operator. The result of a ROLLUP operation can be produced using a number of SELECT statements connected by the UNION operator as shown in Example 2.8. The additional SELECT statements generate subtotals for each ordered subset of grouped columns. With three grouped columns, three additional SELECT statements are needed to generate the subtotals. With *N* grouped columns, *N* ad-

ditional SELECT statements are needed. Obviously, the ROLLUP operator is much easier to write than a large number of additional SELECT statements.

Example 2.8: Rewrite of Example 2.7 without Using the ROLLUP Operator

In each additional SELECT statement, a default value (0 for numeric columns and '' for text columns) replaces the column in which totals are not generated.

```
SELECT StoreZip, TimeYear, TimeMonth, SUM(SalesDollar) AS SumSales
 FROM Sales, Store, TimeDim
 WHERE Sales.StoreId = Store.StoreId
   AND Sales.TimeNo = TimeDim.TimeNo
   AND (StoreNation = 'USA' OR StoreNation = 'Canada')
   AND TimeYear BETWEEN 2012 AND 2013
 GROUP BY StoreZip, TimeYear, TimeMonth
 UNION
SELECT StoreZip, TimeYear, 0, SUM(SalesDollar) AS SumSales
 FROM Sales, Store, TimeDim
 WHERE Sales.StoreId = Store.StoreId
   AND Sales.TimeNo = TimeDim.TimeNo
   AND (StoreNation = 'USA' OR StoreNation = 'Canada')
   AND TimeYear BETWEEN 2012 AND 2013
 GROUP BY StoreZip, TimeYear
UNION
SELECT StoreZip, 0, 0, SUM(SalesDollar) AS SumSales
 FROM Sales, Store, TimeDim
 WHERE Sales.StoreId = Store.StoreId
   AND Sales.TimeNo = TimeDim.TimeNo
   AND (StoreNation = 'USA' OR StoreNation = 'Canada')
   AND TimeYear BETWEEN 2012 AND 2013
 GROUP BY StoreZip
UNION
SELECT '', 0, 0, SUM(SalesDollar) AS SumSales
 FROM Sales, Store, TimeDim
 WHERE Sales.StoreId = Store.StoreId
   AND Sales.TimeNo = TimeDim.No
   AND (StoreNation = 'USA' OR StoreNation = 'Canada')
   AND TimeYear BETWEEN 2012 AND 2013;
```

2.2.3 GROUPING SETS Operator

For more flexibility than is provided by the CUBE and ROLLUP operators, you can use the GROUPING SETS operator. When you use the GROUPING SETS operator, you explicitly specify the combinations of columns for which you need totals. In contrast, the specification of subtotals is implicit in the CUBE and ROLLUP operators. The GROUPING SETS operator is appropriate when precise control over grouping is needed. If explicit control is not required, the CUBE and ROLLUP operators provide more succinct specification.

GROUPING SETS Operator: an operator in the GROUP BY clause that requires explicit specification of column combinations. The GROUPING SETS operator is appropriate when precise control over grouping and subtotals is required.

To depict the GROUPING SETS operator, the previous examples are recast using the GROUPING SETS operator. In Example 2.9, the GROUPING SETS operator involves subtotals for the *StoreZip* and *TimeMonth* columns along with the grand total denoted by the empty parentheses. The subset (*StoreZip*, *TimeMonth*) also must be specified because all column combinations must be explicitly specified, even the normal grouping without the GROUPING SETS opera-

tor. Example 2.10 contains eight column combinations to provide the same result as Example 2.4 with the CUBE of three columns. Example 2.11 contains three column combinations to provide the same result as Example 2.7 with the ROLLUP of three columns.

Example 2.9 (Oracle): GROUP BY Clause Using the GROUPING SETS Operator

This example produces the same result as Example 2.2.

```
SELECT StoreZip, TimeMonth, SUM(SalesDollar) AS SumSales
 FROM Sales, Store, TimeDim
 WHERE Sales.StoreId = Store.StoreId
   AND Sales.TimeNo = TimeDim.TimeNo
   AND (StoreNation = 'USA' OR StoreNation = 'Canada')
   AND TimeYear = 2012
 GROUP BY GROUPING SETS((StoreZip, TimeMonth), StoreZip,
       TimeMonth, ());
```

Example 2.10 (Oracle): GROUP BY Clause Using the GROUPING SETS Operator

This example produces the same result as Example 2.4.

```
SELECT StoreZip, TimeYear, TimeMonth, SUM(SalesDollar) AS SumSales
 FROM Sales, Store, TimeDim
 WHERE Sales.StoreId = Store.StoreId
   AND Sales.TimeNo = TimeDim.TimeNo
   AND (StoreNation = 'USA' OR StoreNation = 'Canada')
   AND TimeYear BETWEEN 2012 AND 2013
 GROUP BY GROUPING SETS((StoreZip, TimeYear, TimeMonth),
    (StoreZip, TimeMonth), (StoreZip, TimeYear),
    (TimeMonth, TimeYear), StoreZip, TimeMonth, TimeYear, () );
```

Example 2.11 (Oracle): GROUP BY Clause Using the GROUPING SETS Operator

This example produces the same result as Example 2.7.

```
SELECT StoreZip, TimeYear, TimeMonth, SUM(SalesDollar) AS SumSales
 FROM Sales, Store, TimeDim
 WHERE Sales.StoreId = Store.StoreId
   AND Sales.TimeNo = TimeDim.TimeNo
   AND (StoreNation = 'USA' OR StoreNation = 'Canada')
   AND TimeYear BETWEEN 2012 AND 2013
 GROUP BY GROUPING SETS((StoreZip, TimeYear, TimeMonth),
     (StoreZip, TimeYear), StoreZip,() );
```

Example 2.12 depicts a situation in which the GROUPING SETS operator is preferred to the CUBE operator. Because the *TimeYear* and *TimeMonth* columns are from the same dimension hierarchy, a full cube usually is not warranted. Instead, the GROUPING SETS operator can be used to specify the column combinations from which subtotals are needed. Subtotals involving *TimeMonth* without *TimeYear* are excluded in Example 2.12 but are included in a full CUBE operation.

Example 2.12 (Oracle): GROUP BY clause using the GROUPING SETS operator to indicate the column combinations from which subtotals are needed

```
SELECT StoreZip, TimeYear, TimeMonth, SUM(SalesDollar) AS SumSales
 FROM Sales, Store, TimeDim
 WHERE Sales.StoreId = Store.StoreId
   AND Sales.TimeNo = TimeDim.TimeNo
   AND (StoreNation = 'USA' OR StoreNation = 'Canada')
   AND TimeYear BETWEEN 2012 AND 2013
 GROUP BY GROUPING SETS((StoreZip, TimeYear, TimeMonth),
    (StoreZip, TimeYear), (TimeYear, TimeMonth),
    StoreZip, TimeYear, () );
```

2.2.4 GROUP BY Operator Variations and Functions for Business Intelligence

The CUBE, ROLLUP, and GROUPING SETS operators can be combined to provide the appropriate mix of explicit grouping specifications with the GROUPING SETS operator along with subtotals provided by the ROLLUP and CUBE operators. This list provides some of the possible variations when using these operators.

- A partial CUBE can be used to produce subtotals for a subset of independent dimensions. For example, the clause `GROUP BY TimeMonth, CUBE(ItemBrand, StoreState)` produces totals on the column subsets <*TimeMonth*, *ItemBrand*, *StoreState*>, <*TimeMonth*, *ItemBrand*>, <*TimeMonth*, *StoreState*>, and <*TimeMonth*>.

- A partial ROLLUP can be done to produce subtotals for a subset of columns from the same dimension hierarchy. For example the clause `GROUP BY ItemBrand, ROLLUP(TimeYear, TimeMonth, TimeDay)` produces totals on the column subsets <*ItemBrand*, *TimeYear*, *TimeMonth, TimeDay*>, <*ItemBrand*, *TimeYear*, *TimeMonth* >, <*ItemBrand*, *TimeYear* >, and <*ItemBrand*>.

- Composite columns can be used with the CUBE or ROLLUP operators to skip some subtotals. For example, the clause `GROUP BY ROLLUP(TimeYear, (TimeQuarter, TimeMonth), TimeDay)` produces totals on the column subsets < *TimeYear*, *TimeQuarter, TimeMonth, TimeDay*>, <*TimeYear*, *TimeQuarter, TimeMonth*>, <*TimeYear*>, and < >. The composite column <*TimeQuarter, TimeMonth*> is treated as a single column in the ROLLUP operation.

- CUBE and ROLLUP operations can be included in a GROUPING SETS operation. For example the clause `GROUP BY GROUPING SETS(ItemBrand, ROLLUP(TimeYear, TimeMonth), StoreState)` produces totals on the column subsets <*ItemBrand*>, <*StoreState*>, <*TimeYear, TimeMonth*>, <*TimeYear*>, and < >. The nested ROLLUP operation creates subtotals on the column subsets <*TimeYear, TimeMonth*>, <*TimeYear*>, and < >.

In addition to the GROUP BY extensions, a number of new functions can be used in the SELECT statement for business intelligence queries. Some of the typical functions are shown in the following list.

- Ranking functions support requests for the top or the bottom percentage of results.

- Ratio functions simplify the formulation of queries that compare individual values to group totals.

- Functions for moving totals and averages allow smoothing of data for time-series analysis. SQL:2011 provides the WINDOW clause to specify moving averages.

To provide additional details, ranking functions in Oracle are reviewed. Understanding ranking functions can improve your ability to support data warehouse users in formulating analytical queries. A ranking function computes the rank or position of a row compared to other rows in a table based on criteria defining a set of values. Table 2-10 summarizes ranking functions available in Oracle. The ranking functions can be used as an aggregate function or as an analytic function. Aggregate functions return a single result row derived from values in a group of rows. For example, the RANK function as an aggregate function can be used to determine the sales ranking for a particular store relative to all stores. Analytic functions return a value for each row in a query result. An analytic function value depends on a group of rows however, not just the values in the same row. For example, the RANK function as an analytical function can be used to determine the sales rank for each row in a subset of stores relative to all stores.

Table 2-10: Summary of Oracle Ranking Functions

Function Keyword	Description
RANK	Calculates the relative position of a value in a group of values; rows with equal values for the ranking criteria receive the same rank; Rank values are skipped so ranks are not consecutive when equal values exist.
DENSE_RANK	Calculates the relative position of a value in a group of values; rows with equal values for the ranking criteria receive the same rank; Rank values are not skipped so ranks are consecutive numbers.
CUME_DIST	Calculates the fraction of values at or below a specified interval
PERCENT_RANK	Similar to CUME_DIST except that it uses rank values instead of counts to calculate the percentage of values
NTILE	Supports calculation of equal-sized divisions such as quartiles and deciles; Each division has an almost equal number of rows.

2.3 Summary Data Storage and Optimization

To support queries involving large fact tables, relational DBMSs provide materialized views. A materialized view is a stored view that must be periodically synchronized with its source data. Materialized views are attractive in data warehouses because the source data is stable except for periodic refreshments that are usually be performed during nonpeak times. In contrast, traditional (nonmaterialized) views dominate operational database processing because the refresh cost can be high. Along with materialized views, most relational DBMSs support automatic substitution of materialized views for source tables in a process known as query rewriting. This section depicts materialized views using the Oracle syntax and provides examples of the query rewriting process.

> **Materialized View**: a stored view that must be periodically synchronized with its source data. Materialized views support storage of summarized data for fast query response.

2.3.1 Materialized Views in Oracle

Specification of a materialized view in Oracle involves elements of base table specification and the mapping specification of traditional views along with the specification of materialization properties. Because materialized views are stored, most of the storage properties for base tables can also be specified for materialized views. Since the storage properties are not the focus here, they will not be depicted. The mapping specification is the same for traditional views as for materialized views. A SELECT statement provides the mapping necessary to populate a materialized view. The materialization properties include

- Method of refresh (incremental or complete): Oracle has a number of restrictions on the types of materialized views that can be incrementally refreshed so incremental refreshment will not be discussed here.
- Refresh timing (on demand or on commit): For the on demand option, Oracle provides the DBMS_MView package with several refresh procedures (Refresh, Refresh_All_MViews, Refresh_Dependent) to specify refresh timing details.

- Build timing (immediate or deferred): For the deferred option, the refresh procedures in the DBMS_MView package can be used to specify the details of populating the materialized view.

Examples 2.13 to 2.15 depict the syntax of the CREATE MATERIALIZED VIEW statement. These statements appear similar to CREATE VIEW statements except for the materialization clauses. The build timing is immediate in Examples 2.13 and 2.15, while the build timing is deferred in Example 2.14. The refresh method is complete and the refresh timing is on demand in all three materialized views. The SELECT statement following the AS keyword provides the mapping to populate a materialized view.

Example 2.13 (Oracle): Materialized View Containing Sales for all Countries for Years after 2010 Grouped by State and Year

```
CREATE MATERIALIZED VIEW MV1
BUILD IMMEDIATE
REFRESH COMPLETE ON DEMAND
ENABLE QUERY REWRITE AS
SELECT StoreState, TimeYear, SUM(SalesDollar) AS SUMDollar1
 FROM Sales, Store, TimeDim
 WHERE Sales.StoreId = Store.StoreId
   AND Sales.TimeNo = TimeDim.TimeNo
   AND TimeYear > 2010
 GROUP BY StoreState, TimeYear;
```

Example 2.14 (Oracle): Materialized View Containing USA Sales in all Years Grouped by State, Year, and Month

```
CREATE MATERIALIZED VIEW MV2
BUILD DEFERRED
REFRESH COMPLETE ON DEMAND
ENABLE QUERY REWRITE AS
SELECT StoreState, TimeYear, TimeMonth,
       SUM(SalesDollar) AS SUMDollar2
 FROM Sales, Store, TimeDim
 WHERE Sales.StoreId = Store.StoreId
   AND Sales.TimeNo = TimeDim.TimeNo
   AND StoreNation = 'USA'
 GROUP BY StoreState, TimeYear, TimeMonth;
```

Example 2.15 (Oracle): Materialized View Containing Canadian Sales before 2011 Grouped by City, Year, and Month

```
CREATE MATERIALIZED VIEW MV3
BUILD IMMEDIATE
REFRESH COMPLETE ON DEMAND
ENABLE QUERY REWRITE AS
SELECT StoreCity, TimeYear, TimeMonth,
       SUM(SalesDollar) AS SUMDollar3
 FROM Sales, Store, TimeDim
 WHERE Sales.StoreId = Store.StoreId
   AND Sales.TimeNo = TimeDim.TimeNo
   AND StoreNation = 'Canada'
   AND TimeYear <= 2010
 GROUP BY StoreCity, TimeYear, TimeMonth;
```

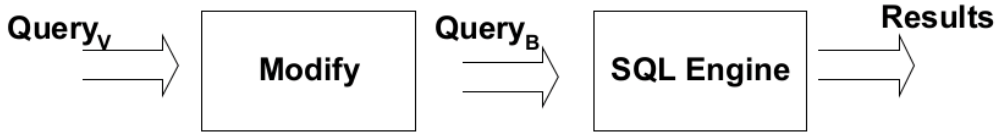
User awareness is another difference between traditional views and materialized views. For queries using operational databases, traditional views are used in place of base tables to simplify query formulation. A user perceives the database as a view shielding the user from the complexities of the base tables. In contrast, for data warehouse queries submitted by users, fact and dimension tables are used although traditional views may also hide the complexity of the data warehouse schema. In addition, the fact and dimension tables may be hidden through a query tool to simplify

query formulation. However, data warehouse users are not aware of materialized views as materialized views are solely performance aids managed by the DBMS. The DBMS substitutes materialized views for base tables to improve performance of queries in a process known as query rewriting. In addition, the DBMS may provide a design tool to help select the best set of materialized views.

2.3.2 Query Rewriting Principles

The query rewriting process for materialized views reverses the query modification process for traditional views. The query modification process (Figure 2.10) substitutes base tables for views so that materialization of the views is not needed. Figure 2.10 shows that a query using a view is modified or rewritten as a query using base tables only; then the modified query is executed using the base tables. For databases with a mixture of retrieval and update activity, modification provides an efficient way to process a query on a view.

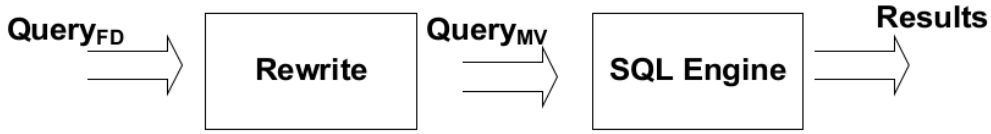
In contrast, the query rewriting process (Figure 2.11) substitutes materialized views for fact and dimension tables to avoid accessing large fact and dimension tables. The substitution process is only performed if performance improvements are expected.



Query$_V$: query that references a view

Query$_B$: modification of Query$_V$ such that references to the view are replaced by references to base tables.

Figure 2.10: Process Flow of View Modification



Query$_{FD}$: query that references fact and dimension tables

Query$_{MV}$: rewrite of Query$_{FD}$ such that materialized views are substituted for fact and dimension tables whenever justified by expected performance improvements.

Figure 2.11: Process Flow of Query Rewriting

Overall, query rewriting is more complex than query modification because query rewriting involves a more complex substitution process and requires the optimizer to evaluate costs. In both processes, the DBMS, not the user, performs the substitution process. In the query rewriting process, the query optimizer must evaluate whether the substitution will improve performance over the original query. In the query modification process, the query optimizer does not compare the cost of the modified query to the original query because modification usually provides substantial cost savings.

Query Rewriting: a substitution process in which a materialized view replaces references to fact and dimension tables in a query. The query optimizer evaluates whether the substitution will improve performance over the original query without the materialized view substitution.

Query Rewriting Details

Query rewriting involves a matching process between a query using fact and dimension tables and a collection of materialized views containing summary data. In brief, a materialized view can provide data for a query if the materialized view matches the row conditions, grouping columns, and aggregate functions as explained below and summarized in Table 2-11.

- Row condition match: materialized view rows specified by its WHERE condition must contain the query rows defined by its WHERE condition. Rewrite is not possible if a materialized view contains more restrictive conditions than a query. Rewrite is possible if the conditions in a query are at least as restrictive as a materialized view. For example, if a materialized view has the conditions StoreNation = 'USA' AND TimeYear = 2012, but a query only has the condition StoreNation = 'USA', the materialized view cannot provide data for the query because the conditions in the materialized view are more restrictive.

- Grouping match for level of detail: Grouping columns in a materialized view must be a superset of the grouping columns in a query. Rewrite is not possible if a grouping column in a query does not appear in a materialized view. Rewrite is possible if a query contains grouping columns that do not appear in a materialized view. For example, a query with a grouping on *TimeYear* and *TimeMonth* cannot use a materialized view with a grouping on *TimeYear*. However, a materialized view with grouping on *TimeYear* and *TimeMonth* can be rolled up to provide data for a query with grouping on *TimeYear*.

- Grouping match for functional dependencies: rewrite is not possible if a query contains grouping columns not in a materialized view unless the columns can be derived by functional dependencies. Functional dependencies are derived from primary keys, candidate keys, and dimension dependencies (via the DETERMINES clause in the Oracle CREATE DIMENSION statement). For example, a query with a grouping on *StoreCity* can be derived from a materialized view with a grouping on *StoreId* because *StoreId* → *StoreCity*. Joins can be used to retrieve columns in a query but not in a materialized view as long as there is a functional relationship (usually a 1-M relationship) involving the tables.

- Aggregate match: aggregates in the query must match available aggregates in the materialized view or be derivable from aggregates in the materialized view. For example, a query containing average is derivable from a materialized view containing sum and count.

Table 2-11: Summary of Matching Requirements for Query Rewriting

Matching Type	Requirements
Row conditions	Materialized view rows specified by its WHERE condition must contain the query rows defined by its WHERE condition.
Grouping detail	Grouping columns in a materialized view must be a superset of the grouping columns in a query.
Grouping dependencies	Query columns must match or be derivable by functional dependencies from materialized view columns.
Aggregate functions	Query aggregate functions must match or be derivable from materialized view aggregate functions.

Example 2.16 presents an example data warehouse query and the rewritten query to depict the matching process. Table 2-12 depicts matching between *MV1* and the query in Example 2.16. *MV1* and the query match directly on the grouping columns and aggregate computations. The condition on *TimeYear* (> 2010) in *MV1* contains the query condition (2012). In addition, the query contains an extra condition on *StoreNation*. The materialized view result does not need

to contain *StoreNation* because *StoreState* → *StoreNation*. Grouping is not necessary in the rewritten query because identical grouping is already performed in *MV1*. The DISTINCT keyword removes duplicate state values in the result because the *Store* table has multiple rows with the same state value.

Example 2.16: Data warehouse Query and Rewritten Query using the *MV1* Materialized View

```
-- Data warehouse query
SELECT StoreState, TimeYear, SUM(SalesDollar)
 FROM Sales, Store, TimeDim
 WHERE Sales.StoreId = Store.StoreId
   AND Sales.TimeNo = TimeDim.TimeNo
   AND StoreNation IN ('USA','Canada')
   AND TimeYear = 2012
 GROUP BY StoreState, TimeYear;

-- Query Rewrite: replace Sales and TimeDim tables with MV1
SELECT DISTINCT MV1.StoreState, TimeYear, SumDollar1
 FROM MV1, Store
 WHERE MV1.StoreState = Store.StoreState
   AND TimeYear = 2012
   AND StoreNation IN ('USA','Canada');
```

Table 2-12: Matching between Materialized View and Example 2.16

	Materialized View	**Query**
Grouping	StoreState, TimeYear	StoreState, TimeYear
Conditions	TimeYear > 2010	TimeYear = 2012 StoreNation = ('USA', 'Canada')
Aggregates	SUM(SalesDollar)	SUM(SalesDollar)

Example 2.17 presents a more complex example of query rewriting involving three SELECT blocks combined using the UNION operator. Table 2-13 depicts matching between the materialized views (*MV1*, *MV2*, *MV3*) and the query in Example 2.2. The first query block retrieves the total sales in the United States or Canada from 2010 to 2013. The second query block retrieves the USA store sales in 2010. The third query block retrieves Canadian store sales in 2010. The GROUP BY clauses are necessary in the second and third query blocks to roll-up the finer level of detail in the materialized views. In the third query block, the condition on *StoreNation* is needed because some cities have identical names in both countries. The materialized view results do not need to contain *StoreNation* because *StoreState* → *StoreNation*. The DISTINCT keyword in the first and third query blocks removes duplicate state values in the result because the *Store* table has multiple rows with the same state value.

Example 2.17: Data Warehouse Query and Rewritten Query using the *MV1*, *MV2*, and *MV3* Materialized Views

```
-- Data warehouse query
SELECT StoreState, TimeYear, SUM(SalesDollar)
 FROM Sales, Store, TimeDim
 WHERE Sales.StoreId = Store.StoreId
   AND Sales.TimeNo = TimeDim.TimeNo
   AND StoreNation IN ('USA','Canada')
   AND TimeYear BETWEEN 2010 and 2013
 GROUP BY StoreState, TimeYear;
```

```
-- Query Rewrite
SELECT DISTINCT MV1.StoreState, TimeYear, SumDollar1 AS StoreSales
 FROM MV1, Store
 WHERE MV1.StoreState = Store.StoreState
   AND TimeYear <= 2013
   AND StoreNation IN ('USA','Canada')
UNION
SELECT StoreState, TimeYear, SUM(SumDollar2) as StoreSales
 FROM MV2
 WHERE TimeYear = 2010
 GROUP BY StoreState, TimeYear
UNION
SELECT DISTINCT StoreState, TimeYear, SUM(SumDollar3) as StoreSales
 FROM MV3, Store
 WHERE MV3.StoreCity = Store.StoreCity
   AND TimeYear = 2010 AND StoreNation = 'Canada'
 GROUP BY StoreState, TimeYear;
```

Table 2-13: Matching between Materialized Views and Example 2.17

	MV1	MV2	MV3	Query
Grouping	StoreState, TimeYear	StoreState, TimeMonth, TimeYear	StoreCity, StoreState, TimeYear	StoreState, TimeYear
Conditions	TimeYear > 2010	StoreNation = 'USA'	TimeYear <= 2010 StoreNation = 'Canada'	TimeYear BETWEEN 2010 AND 2013 StoreNation = ('USA', 'Canada')
Aggregates	SUM(SalesDollar)	SUM(SalesDollar)	SUM(SalesDollar)	SUM(SalesDollar)

Example 2.18 extends Example 2.17 with a CUBE operator. In the rewritten query, the CUBE is performed one time at the end rather than in each SELECT block. To perform the CUBE one time, the FROM clause should contain a nested query. An alternative to the nested query in the FROM clause is to place the nested query in a separate CREATE VIEW statement.

Example 2.18: Data Warehouse Query and Rewritten Query Using the *MV1*, *MV2*, and *MV3* Materialized Views

```
-- Data warehouse query
SELECT StoreState, TimeYear, SUM(SalesDollar)
 FROM Sales, Store, TimeDim
 WHERE Sales.StoreId = Store.StoreId
   AND Sales.TimeNo = TimeDim.TimeNo
   AND StoreNation IN ('USA','Canada')
   AND TimeYear BETWEEN 2010 and 2013
 GROUP BY CUBE(StoreState, TimeYear);

-- Query Rewrite
SELECT StoreState, TimeYear, SUM(StoreSales) as SumStoreSales
 FROM (
  SELECT DISTINCT MV1.StoreState, TimeYear, SumDollar1 AS StoreSales
   FROM MV1, Store
   WHERE MV1.StoreState = Store.StoreState
     AND TimeYear <= 2013
     AND StoreNation IN ('USA','Canada')
 UNION
  SELECT StoreState, TimeYear, SUM(SumDollar2) as StoreSales
   FROM MV2
```

```
    WHERE TimeYear = 2010
    GROUP BY StoreState, TimeYear
  UNION
   SELECT DISTINCT StoreState, TimeYear, SUM(SumDollar3) as StoreSales
    FROM MV3, Store
    WHERE MV3.StoreCity = Store.StoreCity
      AND TimeYear = 2010 AND StoreNation = 'Canada'
    GROUP BY StoreState, TimeYear )
GROUP BY CUBE(StoreState, TimeYear);
```

These examples indicate the range of query rewriting possibilities rather than capabilities of actual DBMSs. Most enterprise DBMSs support query rewriting, but the range of query rewriting supported varies. Because of the complexity and the proprietary nature of query rewrite algorithms, the details of query rewriting algorithms are beyond the scope of this textbook.

2.3.3 Storage and Optimization Technologies

Several storage technologies have been developed to provide multidimensional data capabilities. The storage technologies support On Line Analytic Processing (OLAP), a generic name applied to business intelligence capabilities for data cubes. In addition to specific storage technologies, DBMS vendors have developed data warehouse appliances, complete hardware and software solutions for deploying enterprise level data warehouses. This section describes the features of OLAP storage technologies along with details about vendor offerings of data warehouse appliance products.

MOLAP (Multidimensional OLAP)

Originally, vendors of business intelligence software developed a storage architecture that directly manipulates data cubes. This storage architecture, known as MOLAP for Multidimensional OLAP, was the only choice as a storage technology for data warehouses until the mid-1990s. At the current time, MOLAP has been eclipsed as the primary storage architecture for data warehouses, but it still is an important technology for summary data cubes and small data warehouses and data marts.

MOLAP storage engines directly manipulate stored data cubes. The storage engines of MOLAP systems are optimized for the unique characteristics of multidimensional data such as sparsity and complex aggregation across thousands of cells. Because data cubes are precomputed, MOLAP query performance is generally better than competing approaches that use relational database storage. Even with techniques to deal with sparsity, MOLAP engines can be overwhelmed by the size of data cubes. A fully calculated data cube may expand many times as compared to the raw input data. This data explosion problem limits the size of data cubes that MOLAP engines can manipulate.

MOLAP: a storage engine that directly stores and manipulates data cubes. MOLAP engines generally offer the best query performance but place limits on the size of data cubes.

ROLAP (Relational OLAP)

Because of the potential market size and growth of data warehouse processing, vendors of relational DBMSs have extended their products with additional features to support operations and storage structures for multidimensional data. These product extensions are collectively known as ROLAP for Relational OLAP. Given the growing size of data warehouses and the intensive research and development by relational DBMS vendors, it was only a matter of time before ROLAP became the dominant storage engine for data warehouses.

In the ROLAP approach, relational databases store multidimensional data using the star schema or its variations, as described in Chapter 16. Data cubes are dynamically constructed from fact and dimension tables as well as from materialized views. Typically, only a subset of a data cube must be constructed as specified in a user's query. Extensions

to SQL as described in section 2.2 allow users to manipulate the dimensions and measures in virtual data cubes.

ROLAP: relational DBMS extensions to support multidimensional data. ROLAP engines support a variety of storage and optimization techniques for summary data retrieval.

ROLAP engines incorporate a variety of storage and optimization techniques for summary data retrieval. This list explains the most prominent techniques:

- Bitmap join indexes are particularly useful for columns in dimension tables with few values such as *Cust-State*. For background about bitmap join indexes, you should read Appendix A. A bitmap join index provides a precomputed join from the values of a dimension table to the rows of a fact table. To support snowflake schemas, some DBMS vendors support bitmap join indexes for dimension tables related to other dimension tables. For example, a bitmap index for the *Division.DivManager* column has an index record that contains a bitmap for related rows of the *Store* table and a second bitmap for rows of the *Sales* table related to matching rows of the *Store* table.
- Star join query optimization uses bitmap join indexes on dimension tables to reduce the number of rows in a fact table to retrieve. A star join involves a fact table joined with one or more dimension tables. Star join optimization involves three phases. In the first phase, the bitmap join indexes on each dimension table are combined using the union operator for conditions connected by the OR operator and the intersection operator for conditions connected by the AND operator. In the second phase, the bitmaps resulting from the first phase are combined using the intersection operator. In the third phase, the rows of the fact table are retrieved using the bitmap resulting from the second phase. Star join optimization can result in substantially reduced execution time as compared to traditional join algorithms that combine two tables at a time.
- Query rewriting using materialized views can eliminate the need to access large fact and dimension tables. If materialized views are large, they can be indexed to improve retrieval performance. Query rewriting uses the query optimizer to evaluate the benefit of using materialized views as compared to the fact and dimension tables.
- Summary storage advisors determine the best set of materialized views that should be created and maintained for a given query workload. For consistency with other components, the summary advisor is integrated with the query rewriting component and the query optimizer.
- Partitioning, striping, and parallel query execution provide opportunities to reduce the execution time of data warehouse queries. The choices must be carefully studied so that the use of partitioning and striping support the desired level of parallel query execution.

Despite the intensive research and development on ROLAP storage and optimization techniques, MOLAP engines still provide faster query response time. However, MOLAP storage suffers from limitations in data cube size so that ROLAP storage is necessary for fine-grained data warehouses. In addition, the difference in response time has narrowed so that ROLAP storage may involve only a slight performance penalty if ROLAP storage and optimization techniques are properly utilized.

HOLAP (Hybrid OLAP)

Because of the tradeoffs between MOLAP and ROLAP, a third technology known as HOLAP for Hybrid OLAP has been developed to combine ROLAP and MOLAP. HOLAP allows a data warehouse to be divided between relational storage of fact and dimension tables and multidimensional storage of summary data cubes. When an OLAP query is submitted, the HOLAP system can combine data from the ROLAP managed data and the MOLAP managed data.

HOLAP: a storage engine for data warehouses that combines ROLAP and MOLAP storage engines. HOLAP involves both relational and multidimensional data storage as well as combining data from both relational and multidimensional sources for data cube operations.

Despite the appeal of HOLAP, it has potential disadvantages that may limit its use. First, HOLAP can be more complex than either ROLAP or MOLAP, especially if a DBMS vendor does not provide full HOLAP support. To fully support HOLAP, a DBMS vendor must provide both MOLAP and ROLAP engines as well as tools to combine both storage engines in the design and operation of a data warehouse. Second, there is considerable overlap in functionality between the storage and optimization techniques in ROLAP and MOLAP engines. It is not clear whether the ROLAP storage and optimization techniques should be discarded or used in addition to the MOLAP techniques. Third, because the difference in response time has narrowed between ROLAP and MOLAP, the combination of MOLAP and ROLAP may not provide significant performance improvement to justify the added complexity.

Data Warehouse Appliances

Data warehouse appliances provide prepackaged solutions for operating a data warehouse using various storage technologies and optimization methods. A data warehouse appliance is a combination of hardware and software components for rapid deployment and transparent operation of data warehouses. The components typically include an operating system, a DBMS, server hardware, and storage devices. Early appliances emphasized proprietary components but the trend now is for more open and industry standard components especially commodity hardware and open source operating systems.

> **Data Warehouse Appliance**: a prepackaged solution for operating a data warehouse using various storage technologies and optimization methods. A data warehouse appliance is a combination of hardware and software components for rapid deployment and transparent operation of data warehouses.

Data warehouse appliances offer the promise of increased performance, reduced maintenance costs, and improved scalability. Most appliance products provide parallel database processing and other performance enhancements to improve both query and refresh processing. The performance improvement is partially due to dedicated components and tuning for data warehouse processing without compromise for other types of processing. Vendors have designed data warehouse appliances for rapid deployment and transparent operation. Organizations have reported lower staffing costs for data warehouse administrators due to less performance tuning and monitoring. The performance and maintenance benefits are especially important as data warehouse processing loads increase. Vendors have designed data warehouse appliances for scalability with relatively easy addition of components as loads increase.

Data warehouse appliances have some potential disadvantages. Using a data warehouse appliance often means redeploying a data warehouse to a new environment. Migration costs are a potential problem although data warehouse

appliance vendors provide support to reduce the burden of data migration. Query migration may also be required depending on the level of proprietary SQL used in the current environment. The fixed costs (purchase and lease) of data warehouse appliances can be considerable although these costs must be weighed against cost reductions for improved performance and lower staffing costs. Data warehouse appliance vendors emphasize lower costs of ownership balancing the fixed costs against lower operating costs.

The market for data warehouse appliances provides both complete solutions with both hardware and software components and partial solutions (sometimes known as software appliances) with software components only as shown in Table 2-14. The complete solutions are mostly offered by DBMS vendors sometimes in cooperation with particular hardware vendors. For example, Microsoft provides solutions for both HP and Dell servers packaged with SQL Server. The partial solutions are special purpose analytical engines for data warehouse processing especially data cube operations. Some of the partial solutions use different database models such as column and grid representation along with an emphasis on main memory processing. The partial solutions can be combined with commodity hardware to provide complete solutions for data warehouse processing.

Table 2-14: Major Offerings of Data Warehouse Appliances

Company	Products	Solution Type	Notes
Microsoft	Parallel Data Warehouse, Fast Track Data Warehouse	Complete	Alliances with HP and Dell
IBM	Netezza	Complete	Variety of product offerings for a range of data warehouse needs
TeraData	Teradata Data Warehouse Appliance 2690	Complete	Alliance with Intel
Oracle	Oracle Exadata Intelligent Warehouse Solution	Complete	Variety of configurations for both data warehouse and transaction processing
Vertica	Vertica Analytic Platform	Partial	Columnar data model and analytic processing using commodity hardware
Kognitio	Kognitio Analytical Platform	Partial	In memory analytical engine using commodity hardware

Closing Thoughts

This chapter extended the data warehouse foundation provided in Chapter 1 with detailed coverage of data integration, query language extensions for multidimensional data, and management of summary data. Maintaining a data warehouse is a difficult process that must be carefully managed. This chapter presented the kinds of data sources used in maintaining a data warehouse, a generic workflow describing the tasks involved in maintaining a data warehouse, techniques for data cleaning used in maintaining a data warehouse, features of data integration tools that support data warehouse maintenance activities, and design issues impacting the refresh process. This chapter advocated usage of data integration tools with integrated development environments to reduce the effort to populate and maintain data warehouses.

Because relational DBMSs provide efficient management of large databases, most enterprise data warehouses are implemented using relational databases. For querying relational implementations of data warehouses, extensions to the SQL SELECT statement have been proposed in the SQL standard and implemented by major DBMS vendors. This chapter presented extensions to the GROUP BY clause for subtotal calculations important when summarizing multidimensional data. Details about the CUBE, ROLLUP, and GROUPING SETS operators were shown to extend query formulation skills for multidimensional data in star schemas.

DBMS vendors have made substantial product extensions for efficient management of summary data and new opti-

mization technologies for data warehouse queries. This chapter presented materialized views, a fundamental tool for efficient management of summary data. Details about defining materialized views and the query rewriting process to use materialized views to improve retrieval performance of summary data were provided. This chapter presented storage technologies for data warehouses using both relational and data cube storage engines. Together, background on materialized views and storage technologies equip the student to understand basic issues with efficient implementation of a data warehouse.

Review Concepts

- Kinds of change data used to populate a data warehouse: cooperative, logged, queryable, and snapshot
- Phases in the workflow for maintaining a data warehouse: preparation, integration, and propagation
- Data quality problems encountered in data warehouse loading and maintenance: multiple identifiers, multiple names, different units, missing values, orphaned transactions, multipurpose fields, conflicting data, and different update times
- Regular expressions for decomposing multipurpose text fields into constituent parts
- Metacharacters for specifying patterns in regular expressions involving iteration, range of characters, position, and alteration
- Importance of understanding the reason for missing values in resolving problems
- Entity matching to identify duplicate records when no common, reliable identifier exists
- Entity matching outcomes when comparing records
- Distance measures for string comparisons as fundamental tools for entity matching
- Features of data integration tools: integrated development environments, graphical specification of workflows, non procedural specification of data cleaning tasks, data profiling, change data capture, repository, database connectivity, and job management
- ETL architecture emphasizing the usage of an ETL engine independent of a DBMS engine for performng complex data transformations before loading into data warehouse tables
- ELT architecture emphasizing the usage of a relational DBMS to perform complex data transformations after loading data into data warehouse tables
- Importance of data integration tools to reduce the coding in procedures to maintain a data warehouse
- Control over valid time lag and load time lag in refreshing a data warehouse
- Determining the refresh frequency for a data warehouse: balancing the frequency of refresh against the cost of refresh while satisfying refresh constraints
- Types of refresh constraints: source access, integration, completeness/consistency, availability
- Extensions of the GROUP BY clause for subtotal calculation: CUBE, ROLLUP, and GROUPING SETS operators
- CUBE operator for generating all possible combinations of subtotals
- ROLLUP operator for generating subtotals for coarser parts of a dimension hierarchy
- GROUPING SETS operator for precise control of subtotals
- Equivalence of GROUP BY operators to UNION operations with GROUP BY clauses

- Materialized views for storage of precomputed summary data
- CREATE MATERIALIZED VIEW statement involving method of refresh, refresh timing, build timing, and a SELECT statement
- Process flow for query rewriting (materialized views) versus process flow for view modification (traditional views)
- Query rewriting involving substitution of materialized views for fact and dimension tables to improve performance of data warehouse queries
- Matching requirements for query rewriting involving row conditions, grouping detail, grouping dependencies, and aggregate functions
- Multidimensional storage architectures: ROLAP, MOLAP, and HOLAP
- Star join optimization using bitmap indexes on dimension tables to reduce the number of rows in a fact table to retrieve
- Data warehouse appliances providing combinations of hardware and software components for rapid deployment and transparent operation of data warehouses

Questions

1. What is cooperative change data?
2. What is logged change data?
3. What is queryable change data?
4. What is snapshot change data?
5. Briefly describe the phases of data warehouse maintenance.
6. Briefly define common data quality problems that should be resolved by the preparation and integration phases.
7. Compare the initial loading process and the refresh process for data warehouses.
8. What are the elements of a regular expression?
9. How are regular expressions useful for resolving data quality problems?
10. Briefly explain the iteration metacharacters.
11. Briefly explain the position metacharacters.
12. When should you use an escape character in a regular expression?
13. Why are groups useful in regular expressions?
14. Why is it important to understand the reasons for missing values in resolving data quality problems?
15. What is entity matching?
16. Briefly explain the outcomes of entity matching.
17. Which outcome is typically most important to avoid in entity matching?
18. Why are distance measures for string comparisons important in entity matching?
19. Briefly explain important features of data integration tools.

20. Briefly explain the ETL architecture for data integration.
21. Briefly explain the ELT architecture for data integration.
22. Which data integration architecture (ETL or ELT) will likely dominate in the future?
23. What is valid time lag?
24. What is load time lag?
25. What is the primary objective in managing the refresh process for a data warehouse?
26. What types of constraints affect the refresh process?
27. What is the purpose of the SQL CUBE operator?
28. What is the purpose of the SQL ROLLUP operator?
29. What is the purpose of the SQL GROUPING SETS operator?
30. Briefly list some of the variations of the CUBE, ROLLUP, and GROUPING SETS operators.
31. Why are materialized views important for data warehouses but typically not important for operational databases?
32. What materialization properties does Oracle provide for materialized views?
33. Compare and contrast query rewriting for materialized views to query modification for traditional (nonmaterialized) views.
34. Briefly explain the matching processes to enable query rewriting.
35. Explain the importance of indexing fact and dimension tables in a data warehouse.
36. What are the pros and cons of a MOLAP storage engine?
37. What are the pros and cons of a ROLAP storage engine?
38. What are the pros and cons of a HOLAP storage engine?
39. List some storage and optimization techniques in ROLAP engines.
40. What is star join optimization?
41. What are the advantages of data warehouse appliances?
42. What are possible disadvantages of data warehouse appliances?
43. What are the differences between complete and partial data warehouse appliances?

Problems

The problems provide practice with relational database manipulation of multidimensional data using the store sales snowflake schema introduced in Chapter 1. For your reference, Figure 2.P1 displays the ERD for the store sales snowflake schema. To support the usage of Oracle with these problems, the student section of the textbook's website contains Oracle CREATE TABLE statements and sample data for the tables of the store sales schema.

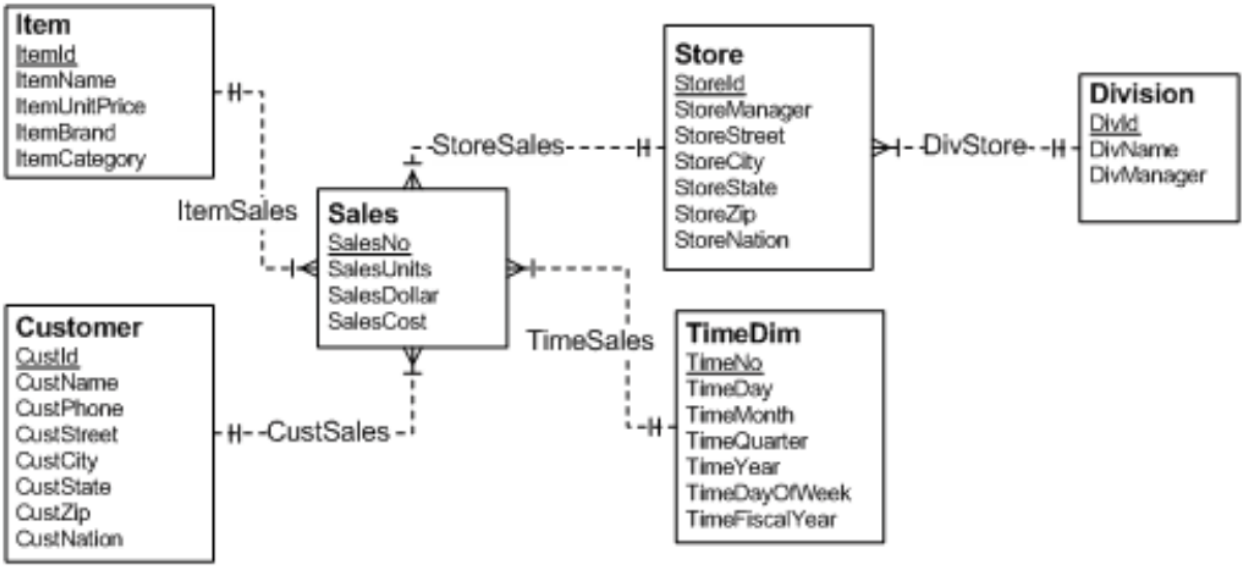


Figure 2.P1: ERD Snowflake Schema for the Store Sales Example

1. Write a SELECT statement to summarize sales by store state, year, and item brand. The result should compute the SUM of the dollar sales for the years 2012 and 2013. The result should include full totals for every combination of grouped fields.

2. Write a SELECT statement to summarize sales by year, quarter, and month. The result should compute the SUM of the dollar sales for the years 2012 and 2013. The result should include partial totals in order of the grouped fields (year, quarter, and month).

3. Write a SELECT statement to summarize sales by store state, month, and year. The result should compute the SUM of the dollar sales for the years 2012 and 2013. The result should include partial totals in order of the hierarchical dimension (year and month). Do not use the GROUPING SETS operator in your SQL statement.

4. Write a SELECT statement to summarize sales by customer state, customer zip, year, and quarter. The result should compute the SUM of the dollar sales for the years 2012 and 2013. The result should include partial totals for hierarchical dimensions (year/quarter and state/zip). Do not use the GROUPING SETS operator in your SQL statement.

5. Rewrite the SQL statement solution for problem 1 without using the CUBE, ROLLUP, or GROUPING SETS operators. To reduce your time, you can write the first few query blocks and then indicate the pattern to rewrite the remaining query blocks. In rewriting the queries, you may use two single quotes (with nothing inside) as the default text value and 0 as the default numeric value.

6. Rewrite the SQL statement solution for problem 2 without using the CUBE, ROLLUP, or GROUPING SETS operators. In rewriting the queries, you may use two single quotes (with nothing inside) as the default text value and 0 as the default numeric value.

7. Rewrite the SQL statement solution for problem 3 without using the CUBE, ROLLUP, or GROUPING SETS operators. In rewriting the queries, you can use two single quotes (with nothing inside) as the default text value and 0 as the default numeric value.

8. Rewrite the SQL statement solution for problem 3 using the GROUPING SETS operator instead of the ROLLUP operator.

9. Rewrite the SQL statement solution for problem 4 using the GROUPING SETS operator instead of the ROLLUP operator.

10. Perform the indicated calculation and show the underlying formula for the following problems. The number of unique values in each dimension is shown in parentheses.

    - Calculate the maximum number of rows for a query with a rollup of year (2), quarter (4), and month (12). Separate the calculation to show the number of rows appearing in the normal GROUP BY result and the number of subtotal rows generated by the ROLLUP operator.

    - Calculate the maximum number of rows in a query with a rollup of year (2), quarter (4), month (12), and weeks per month (4). Separate the calculation to show the maximum number of rows appearing in the normal GROUP BY result and the maximum number of subtotal rows generated by the rollup operator.

    - Calculate the maximum number of rows in a query with a cube of state (5), brands (10), and year (2). Separate the calculation to show the maximum number of rows appearing in the normal GROUP BY result and the maximum number of subtotal rows generated by the cube operator.

    - Calculate the number of subtotal groups in a query with a cube of state (5), division (4), brand (10), and year (2). A subtotal group is equivalent to a SELECT statement when formulating the query without any GROUP BY operators.

11. Write an Oracle CREATE DIMENSION statement for a customer dimension consisting of the customer identifier, the city, the state, the zip, and the country. Define two hierarchies grouping the customer identifier with the city, the state, and the nation and the customer identifier with the zip, the state, and the nation. You should review the material in Chapter 16.3.5 about the Oracle CREATE DIMENSION statement.

12. Write an Oracle CREATE DIMENSION statement for a time dimension consisting of the time identifier, the day, the month, the quarter, the year, the fiscal year, and the day of the week. Define three hierarchies grouping the time identifier, the day, the month, the quarter, and the year, the time identifier and the fiscal year, and the time identifier and the day of the week. You should review the material in Chapter 16.3.5 about the Oracle CREATE DIMENSION statement.

13. Write an Oracle CREATE MATERIALIZED VIEW statement to support the store sales schema. The materialized view should include the sum of the dollar sales and the sum of the cost of sales. The materialized view should summarize the measures by the store identifier, the item identifier, and the time number. The materialized view should include sales in the year 2012.

14. Write an Oracle CREATE MATERIALIZED VIEW statement to support the store sales schema. The materialized view should include the sum of the dollar sales and the sum of the cost of sales. The materialized view should summarize the measures by the store identifier, the item identifier, and the time number. The materialized view should include sales in the year 2013.

15. Rewrite the SQL statement solution for problem 1 using the materialized views in problems 13 and 14. You should ignore the CUBE operator in the solution for problem 1. Your SELECT statement should reference the materialized views as well as base tables if needed.

16. Rewrite the SQL statement solution for problem 1 using the materialized views in problems 13 and 14. Your SELECT statement should reference the materialized views as well as base tables if needed. You should think carefully about how to handle the CUBE operator in your rewritten query.

17. Rewrite the SQL statement solution for problem 3 using the materialized views in problems 13 and 14. You should ignore the ROLLUP operator in the solution for problem 3. Your SELECT statement should reference the materialized views as well as base tables if needed.

18. Rewrite the SQL statement solution for problem 3 using the materialized views in problems 13 and 14. Your SELECT statement should reference the materialized views as well as base tables if needed. You should think carefully about how to handle the ROLLUP operator in your rewritten query.

19. Write an Oracle CREATE MATERIALIZED VIEW statement to support the store sales schema. The materialized view should include the sum of sales units and the sum of the cost of sales. The materialized view should summarize the measures by the customer zip code and year of sales. The materialized view should include sales

in 2012 and before.

20. Write an Oracle CREATE MATERIALIZED VIEW statement to support the store sales schema. The materialized view should include the sum of sales units and the sum of the cost of sales. The materialized view should summarize the measures by the customer zip code, sales year, and sales quarter. The materialized view should include USA sales only.

21. Write an Oracle CREATE MATERIALIZED VIEW statement to support the store sales schema. The materialized view should include the sum of the sales units and sum of the cost of sales. The materialized view should summarize the measures by the customer zip code, sales year, and sales quarter. The materialized view should include Canadian sales for 2012 and 2013.

22. Write a SELECT statement using the base data warehouse tables to retrieve the sum of the sales cost divided by the sum of the unit sales in the USA and Canada in 2012. The result should include customer zip code, year, and sum of the sales cost per unit. Rewrite the SELECT statement using one or more materialized views defined in problems 19 to 21.

23. Write a SELECT statement using the base data warehouse tables to retrieve the sum of the sales cost divided by the sum of the unit sales in the USA and Canada from 2011 to 2013. The result should include customer zip code, year, and sum of the sales cost per unit. Rewrite the SELECT statement using one or more materialized views defined in problems 19 to 21.

24. Identify the subtotals generated with the following GROUP BY clause: `GROUP BY TimeYear, CUBE(ItemCategory, StoreZip).`

25. Identify the subtotals generated with the following GROUP BY clause: `GROUP BY TimeYear, TimeQuarter, CUBE(ItemCategory, StoreZip).`

26. Identify the subtotals generated with the following GROUP BY clause: `GROUP BY ItemCategory, ROLLUP(TimeYear, TimeQuarter, TimeMonth).`

27. Identify the subtotals generated with the following GROUP BY clause: `GROUP BY ItemCategory, StoreZip, ROLLUP(TimeYear, TimeQuarter, TimeMonth).`

28. Identify the subtotals generated with the following GROUP BY clause: `GROUP BY CUBE(ItemCategory, StoreZip), ROLLUP(TimeYear, TimeQuarter, TimeMonth).`

29. Identify the subtotals generated with the following GROUP BY clause: `GROUP BY ROLLUP(ItemCategory, ItemBrand), ROLLUP(TimeYear, TimeQuarter, TimeMonth).`

30. Identify the subtotals generated with the following GROUP BY clause: `GROUP BY ROLLUP(StoreNation, StoreState, (StoreZip, StoreCity)).`

31. Identify the subtotals generated with the following GROUP BY clause: `GROUP BY GROUPING SETS(ItemCategory, ROLLUP(TimeYear, TimeQuarter), StoreZip).`

32. Identify the subtotals generated with the following GROUP BY clause: `GROUP BY GROUPING SETS(TimeYear, StoreZip, CUBE(ItemBrand, ItemCategory), CustZip).`

33. Determine the results of pattern matching using the regular expressions and search strings in Table 2-P1. You should determine the results yourself using knowledge of the metacharacters and then use a regular expression testing tool to verify your predicted results.

Table 2-P1: Regular Expressions and Search Strings to Evaluate

Regular Expression	Search Strings	Evaluation (please complete)
“labou?r”	“pro labor”, “labour union”	

“help*”	“hello”, “hellcat”, “herself”
“help+”	“hepless”, “helpless”, “help me”
“[aeiou][bcdfgm]”	“a dog”, “ihop”, “uomo”
“[A-Z]{1}:[0-9]{2,4}”	“a:100”, “big A:2456”
“^big”	“bill dig”, “one big bill”
“big$”	“bill dig”, “dig bin”
“[^a-z]+[^A-Z]”	“123ab”, “aBCd”, “aaBB”
“^\^”	“^abc”, “abc^”, “abc”
“\$$”	“abc$”, “$abc”, “abc$d”

34. Write a regular expression to match against phone numbers with the format “(ddd) ddd-dddd” where d is a digit and the parentheses are literal characters. The first digit in a phone number should not be a 0. There should not be any other characters at the beginning or ending of the string. Use a regular expression testing tool to test your regular expression.

35. Write a regular expression to match against postal codes with the following format: “ddddd-dddd” where *d* is a digit. The hyphen is a literal character. Both the hyphen and last four digits are optional. If there is a hyphen, the last four digits must be present. There should not be any other characters at the beginning or ending of the string. Use a regular expression testing tool to test your regular expression.

Appendix A Bitmap Indexes

Many DBMSs support bitmap indexes for fast execution of queries involving columns with few values. Figure A.1 depicts a bitmap column index for a sample *Faculty* table. A bitmap contains a string of bits (0 or 1 values) with one bit for each row of a table. In Figure A.1, the length of the bitmap is 12 positions because there are 12 rows in the sample *Faculty* table. A record of a bitmap column index contains a column value and a bitmap. A 0 value in a bitmap indicates that the associated row does not have the column value. A 1 value indicates that the associated row has the column value. The DBMS provides an efficient way to convert a position in a bitmap to a row identifier.

Faculty Table

RowId	FacNo	…	FacRank
1	098-55-1234		Asst
2	123-45-6789		Asst
3	456-89-1243		Assc
4	111-09-0245		Prof
5	931-99-2034		Asst
6	998-00-1245		Prof
7	287-44-3341		Assc
8	230-21-9432		Asst
9	321-44-5588		Prof
10	443-22-3356		Assc
11	559-87-3211		Prof
12	220-44-5688		Asst

Bitmap Column Index on FacRank

FacRank	Bitmap
Asst	110010010001
Assc	001000100100
Prof	000101001010

Figure A.1: Sample Faculty Table and Bitmap Column Index on FacRank

A variation of the bitmap column index is the bitmap join index. In a bitmap join index, the bitmap identifies rows of a related table, not the table containing the indexed column. Thus, a bitmap join index represents a precomputed join from a column in a parent table to the rows of a child table that join with rows of the parent table.

A join bitmap index can be defined for a join column such as *FacNo* or a nonjoin column such as *FacRank*. Figure A.2 depicts a bitmap join index for the *FacRank* column in the *Faculty* table to the rows in the sample *Offering* table. The length of the bitmap is 24 bits because there are 24 rows in the sample *Offering* table. A 1 value in a bitmap indicates that a parent row containing the column value joins with the child table in the specified bit position. For example, a 1 in the first bit position of the Asst row of the join index means that a *Faculty* row with the Asst value joins with the first row of the *Offering* table.

Offering Table

RowId	OfferNo	…	FacNo
1	1111		098-55-1234
2	1234		123-45-6789
3	1345		456-89-1243
4	1599		111-09-0245
5	1807		931-99-2034
6	1944		998-00-1245
7	2100		287-44-3341
8	2200		230-21-9432
9	2301		321-44-5588
10	2487		443-22-3356
11	2500		559-87-3211
12	2600		220-44-5688
13	2703		098-55-1234
14	2801		123-45-6789
15	2944		456-89-1243
16	3100		111-09-0245
17	3200		931-99-2034
18	3258		998-00-1245
19	3302		287-44-3341
20	3901		230-21-9432
21	4001		321-44-5588
22	4205		443-22-3356
23	4301		559-87-3211
24	4455		220-44-5688

Bitmap Join Index on FacRank

FacRank	Bitmap
Asst	11001001000111001001000 1
Assc	00100010010000100010010 0
Prof	00010100101000010100101 0

Figure A.2: Sample Offering Table and Bitmap Join Index on FacRank

Bitmap Index: a secondary file structure consisting of a column value and a bitmap. A bitmap contains one bit position for each row of a referenced table. A bitmap column index references the rows containing the column value. A bitmap join index references the rows of a child table that join with rows of the parent table containing the column value. Bitmap indexes work well for stable columns with few values, typical of tables in a data warehouse.

Bitmap indexes work well for stable columns with few values. The *FacRank* column would be attractive for a bitmap column index because it contains few values and faculty members do not change rank often. The size of the bitmap is not an important issue because compression techniques can reduce the size significantly. Due to the requirement for stable columns, bitmap indexes are most common for data warehouse tables especially as join indexes.

References for Further Study

Several references provide additional details about important parts of Chapter 2. You should consult Bouzeghoub et al. (1999) and Fisher and Berndt (2001) about the refresh process. For details about Oracle data warehousing features, you should consult the online documentation in the Oracle Technology Network (http://www.oracle.com/technetwork). For more details about Talend Open Studio including a free download, visit the Talend website (www.talend.com). For additional details about regular expressions, you should search under “regular expression tutorials” and “regular expression testers”.

Chapter 3

Data Warehouse Administration

Learning Objectives

This chapter provides detailed coverage about the responsibilities and processes of data specialists involved with administration of data warehouses. After this chapter, the student should have acquired the following knowledge and skills:

- Compare and contrast the responsibilities of database administrators and data administrators especially for meeting challenges of big data
- Explain the motivation for data governance and the components of data governance programs in organizations
- Understand the process to select and evaluate DBMSs

Overview

This chapter presents the organizational context for databases and data warehouses along with important processes that data specialists may use to manage data warehouses. This chapter first discusses an organizational context for data warehouses. You will learn about database support for management decision making, the goals of programs to support information management in organizations, the responsibilities of data and database administrators, and the challenges in managing exploding data growth. After explaining the organizational context, this chapter presents processes to manage data warehouses. You will learn processes for data governance and selection/evaluation of complex products such as DBMSs and data integration tools.

3.1 Organizational Context for Managing Data Warehouses

This section reviews management decision-making levels and discusses database support for decision making at all levels. After this background, this section describes organizational programs (information resource management, knowledge management, and data governance), responsibilities of data specialists to manage information resources, and challenges of managing exploding data growth.

3.1.1 Database Support for Management Decision Making

Databases support business operations and management decision making at various levels. Most large organizations have developed many operational databases to help conduct business efficiently. Operational databases directly support major functions such as order processing, manufacturing, accounts payable, and product distribution. The reasons for investing in an operational database are typically faster processing, larger volumes of business, and reduced personnel costs.

Operational Database: a database to support the daily functions of an organization.

As organizations achieve improved operations, they begin to realize the decision-making potential of their databases. Operational databases provide the raw materials for management decision making as depicted in Figure 3.1. Lower-level management can obtain exception and problem reports directly from operational databases. However, much value must be added to leverage the operational databases for middle and upper management. The operational databases must be cleaned, integrated, and summarized to provide value for tactical and strategic decision making. Integration is necessary because operational databases often are developed in isolation without regard for the information needs of tactical and strategic decision making.

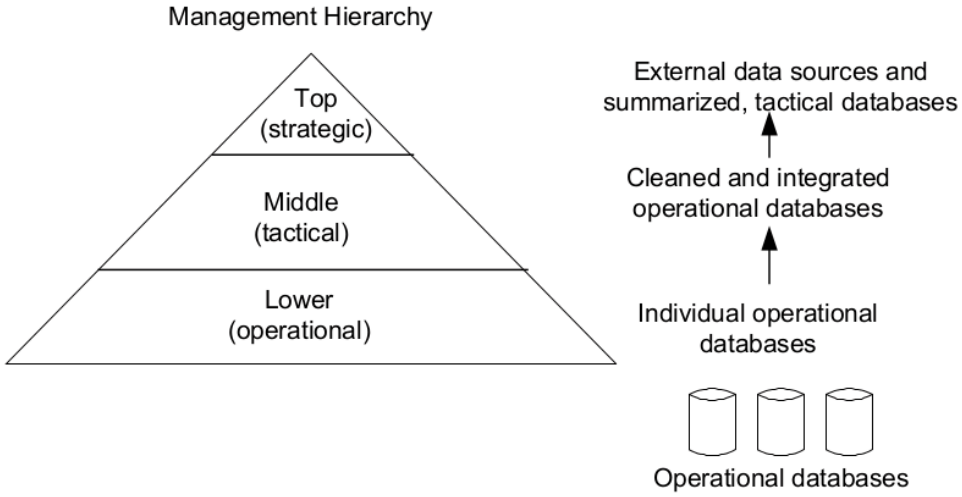


Figure 3.1: Database Support for Management Levels

Table 3-1 provides examples of management decisions and data requirements. Lower-level management deals with short-term problems related to individual transactions. Periodic summaries of operational databases and exception reports assist operational management. Middle management relies on summarized data that are integrated across operational databases. Middle management may want to integrate data across different departments, manufacturing plants, and retail stores. Top management relies on the results of middle management analysis and external data sources. Top management needs to integrate data so that customers, products, suppliers, and other important entities can be tracked across an entire organization. In addition, external data must be captured and then integrated with internal data.

Table 3-1: Examples of Management Decision Making

Level	Example Decisions	Data Requirements
Top	Identify new markets and products; plan growth; reallocate resources across divisions	Economic and technology forecasts; news summaries; industry reports; medium-term performance reports
Middle	Choose suppliers; forecast sales, inventory, and cash; revise staffing levels; prepare budgets	Historical trends; supplier performance; critical path analysis; short-term and medium-term plans
Lower	Schedule employees; correct order delays; find production bottlenecks; monitor resource usage	Problem reports; exception reports; employee schedules; daily production results; inventory levels

3.1.2 Approaches for Managing Data Resources

As a response to the challenges of leveraging operational databases and information technology for management decision making, several management approaches have been developed over the last two decades. The original approach known as information resource management was developed in the 1990s. Information resource management involves processing, distributing, and integrating information throughout an organization. A key element of information resource management is control of information life cycles (Figure 3.2). Each level of management decision making and

business operations has its own information life cycle. For effective decision making, the life cycles must be integrated to provide timely and consistent information. For example, information life cycles for operations provide input to life cycles for management decision making.

Information Life Cycle: the stages of information transformation in an organization. Each entity has its own information life cycle that should be managed and integrated with the life cycles of other entities.

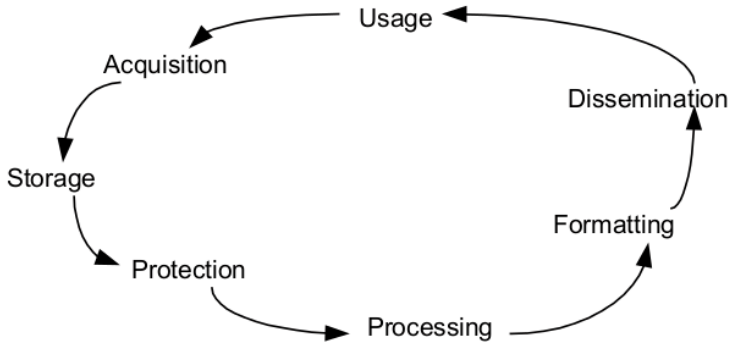


Figure 3.2: Typical Stages of an Information Life Cycle

Data quality is a particular concern for information resource management because of the impact of data quality on management decision making. Data quality involves a number of dimensions such as correctness, timeliness, consistency, completeness, and reliability. Often the level of data quality that suffices for business operations may be insufficient for decision making at upper levels of management. This conflict is especially true for the consistency dimension. For example, inconsistency of customer identification across operational databases can impair decision making at the upper management level. Information resource management emphasizes a long-term, organization-wide perspective on data quality to ensure support for management decision making.

Starting about the mid-1990s, a movement developed to extend information resource management into knowledge management. Traditionally, information resource management has emphasized technology to support predefined recipes for decision making rather than the ability to react to a constantly changing business environment. To succeed in today's business environment, organizations must emphasize fast response and adaptation to extend planning efforts. To meet this challenge, organizations should develop systems that facilitate knowledge creation rather than information management. For knowledge creation, a greater emphasis is on human information processing and organization dynamics to balance the technology emphasis, as shown in Figure 3.3.

Knowledge Management: applying information technology with human information processing capabilities and organization processes to support rapid adaptation to change.

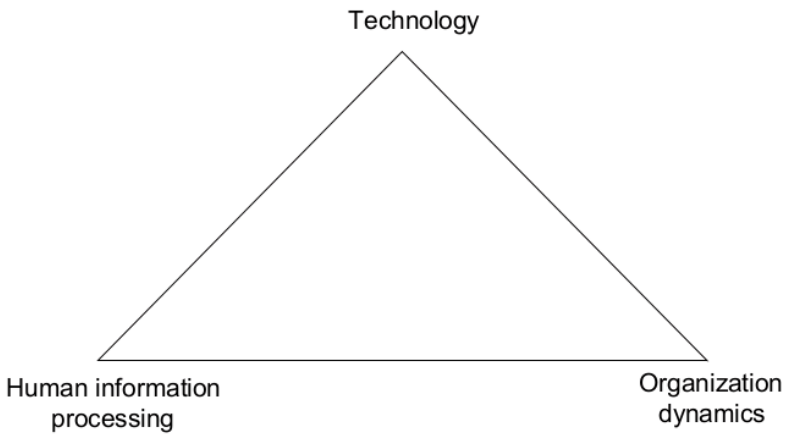


Figure 3.3: Three Pillars of Knowledge Management

This vision for knowledge management provides a context for usage of information technology to solve business problems. The best information technology will fail if not aligned with the human and organization elements. Information technology should amplify individual intellectual capacity, compensate for limitations in human processing, and support positive organization dynamics.

The emphasis on information and knowledge management has shifted to data governance over the last decade. The rapid growth of electronic commerce and financial scandals in the 2000s propelled major changes in regulatory oversight and corporate responsibilities. In the U.S., the Sarbanes-Oxley law, the Health Insurance Portability and Accountability Act, and the more recent Dodd-Frank law have added new corporate responsibilities for data management. The European Union has enacted broad data privacy directives impacting organizations with any European operations.

This changed environment has spurred the movement for data governance. According to the Data Governance Institute (http://www.datagovernance.com), "data governance is the exercise of decision-making and authority for data-related matters." Data governance attempts to mitigate risks associated with the complex regulatory environment, information security, and information privacy especially for personal identifiable data and related business transactions.

Data Governance: according to the Data Governance Institute, data governance involves the application of decision-making and authority for data-related issues.

Data governance provides a system of checks and balances to develop data rules and policies, support application of data rules and policies, and evaluate compliance of data rules and policies as depicted in Figure 3.4. The system of data governance operates in a manner similar to separate government branches in which the legislative branch makes laws, the executive branch enforces laws, and the judicial branch resolves disputes about the meaning and application of laws.

Data governance has been applied to a number of corporate initiatives. Perceived shortcomings in data quality especially in attributes impacted by regulations are primary drivers of data governance initiatives. For data quality improvements, data governance initiatives typically focus on development of data quality measures, reporting status of data quality, and establishing decision rights and accountabilities. Mergers and acquisitions often trigger data governance initiatives to ensure consistent data definitions and integrate corporate policies involving data privacy and security. Similarly, business intelligence developments often lead to data governance initiatives to establish policies for data integration particularly for changes to source data in different parts of an organization.

Figure 3.4: Data Governance Checks and Balances

3.1.3 Responsibilities of Data Specialists

As part of controlling information resources, new management responsibilities have arisen. The data administrator (DA) is a middle- or upper-management position with broad responsibilities for information resource management. The database administrator (DBA) is a support role with responsibilities related to individual databases and DBMSs. Table 3-2 compares the responsibilities of data administrators and database administrators. The data administrator views the information resource in a broader context than the database administrator. The data administrator considers all kinds of data whether stored in relational databases, files, Web pages, or external sources. The data administrator also supports data governance through membership in the data governance office and consulting on activities managed by the data governance office. The database administrator typically considers only data stored in databases and implementing controls to support data governance policies.

Table 3-2: Responsibilities of Data Administrators and Database Administrators

Position	Responsibilities
Data administrator	Develops an enterprise data model
	Establishes inter-database standards and policies about naming, data sharing, and data ownership
	Negotiates contractual terms with information technology vendors
	Develops long-range plans for information technology
	Supports data governance activities
Database administrator	Develops detailed knowledge of individual DBMSs
	Consults on application development
	Performs data modeling, logical database design, and physical database design
	Enforces data administration standards
	Monitors database performance
	Performs technical evaluation of DBMSs
	Creates security, integrity, and rule-processing statements
	Devises standards and policies related to individual databases and DBMSs

Large organizations may offer much specialization in data administration and database administration. For data administration, specialization can occur by task and environment. On the task side, data administrators can specialize in planning versus policy establishment. On the environment side, data administrators can specialize in environments such as business intelligence, operations, and nontraditional data such as images, text, and video. For database administration, specialization can occur by DBMS, task, and environment. Because of the complexities of learning a DBMS, DBAs typically specialize in one product. Task specialization is usually divided between data modeling, application development support, and performance evaluation. Environment specialization is usually divided between transaction processing and data warehouses.

In large organizations, various titles are used for database specialists. The following list explains some common titles used in large organizations.

- Database architect: primarily specializes in data modeling and logical database design
- System DBA: interfaces with system administration and analyzes database impact on hardware and operating system
- Application DBA: specializes in management and usage of procedural objects including triggers, stored procedures, and transaction design
- Senior DBA: a highly experienced DBA who supervises junior DBAs and provides expert trouble shooting
- Performance DBA: specializes in physical database design and performance tuning
- Data warehouse administrator: specializes in operation and development of data warehouses

In small organizations, the boundary between data administration and database administration is fluid. There may not be separate positions for data administrators and database administrators. The same person may perform duties from both positions. As organizations grow, specialization usually develops so that separate positions are created.

Data specialists have many responsibilities for data warehouses, as listed in Table 3-3. Data administrators may perform planning responsibilities involving the data warehouse architecture and the enterprise data model. Database administrators usually perform the more detailed tasks such as performance monitoring and consulting. To support a

large data warehouse, additional software products separate from a DBMS may be necessary such as data integration software. A selection and evaluation process should be conducted to choose the most appropriate products.

Table 3-3: Responsibilities of Database Specialists for Data Warehouses

Area	Responsibilities
Data warehouse usage	Educate and consult about application design and DBMS features for data warehouse processing
Performance monitoring	Monitor data warehouse refresh and query performance; troubleshoot integrity problems; modify resource levels to improve performance
Data warehouse refresh	Determine the frequency of refreshing the data warehouse and the schedule of activities to refresh the data warehouse; design and implement data integration procedures
Data warehouse architecture	Determine architecture to support decision-making needs; select database and data integration products to support architecture; determine resource levels for efficient processing
Enterprise data model	Provide expertise about operational database content; design conceptual data model for the data warehouse; promote data quality to support data warehouse development

3.1.4 Challenges of Big Data

In many organizations, data specialists confront the problem of exploding data growth. According to a 2011 Gartner press release, the amount of worldwide data will grow 59 percent annually over the coming years. A 2011 study by the McKinsey Global Institute[1] projects a 40 percent growth in global data but only a 5 percent growth in global spending on information technology. The growth in data comes from a variety of sources such as sensors in smart phones, energy meters, and automobiles, interaction of individuals in social media websites, radio frequency identification tags in retail, and digitized multimedia content in medicine, entertainment, and security. This data growth breaks existing systems and business processes providing challenges and opportunities for both vendors developing database technology and organizations using database technology.

The phenomenon of explosive data growth known as *big data* was first stated by Doug Laney of the Meta Group in 2001[2]. According to Laney's report, big data contains three dimensions: volume (amount of data), velocity (rate of generating and processing data), and variety (type of data especially the distinction between structured and unstructured data). Most attention is focused on data volumes but the other two dimensions must be managed effectively to deal with problems of big data. Complementing the dimensions of big data, the McKinsey Global Institute defines big data as "datasets whose size is beyond the ability of typical database software to capture, store, manage, and analyze." This definition provides flexibility to vary volumes considered as big data by technology, industry sector, and time.

Big Data: the phenomenon of exploding data growth. Big data has three dimensions, volume, velocity, and variety. The volume of big data that exceeds the limits of database software depends on technology, industry sector, and time.

Big data creates opportunities if managed well. In the previously referenced 2011 report, the McKinsey Global Institute provides a number of startling sources of potential value for big data: $300 billion to the US health care industry resulting in 8 percent reduced costs, €250 billion to the European Union's public sector, $600 billion annual consumer surplus from using personal location data globally, and 60 percent increase in operating margins of retailers. Organizations can unlock the value of big data through matching its increasing velocity, improving accuracy of forecasts and performance of business units, narrowing the segmentation of customers, improving decision making through analytics, and developing new generations of products and services.

1 McKinsey Global Institute, "Big data: The next frontier for innovation, competition, and productivity, May 2011.
2 Laney, Doug, "3D Data Management: Controlling Data Volume, Velocity and Variety," META Group (now Gartner), February 2001.

To effectively manage big data, data specialists should have a clear understanding of data volume units. Basic units of data are the byte (one character or 8 bits), kilobyte (KB), megabyte (MB), and gigabyte (GB). A kilobyte is either 1,024 bytes (binary system measure) or 1,000 bytes (metric system measure). Since computers are binary machines, kilo means 1,024 for digital storage. Storage manufacturers often use the metric system measure perhaps because the public does not understand binary measures. However using the metric system measure provides less capacity than the binary system measure. Thus, a megabyte denotes either 1,000 or 1,024 KB while a gigabyte denotes either 1,000 or 1,024 MB.

Table 3-4 extends the basic data units to larger units along with big data examples. Terabyte was big data in previous decades. Now, it is the typical capacity of hard drives on personal computers. Petabyte is yesterday's big data as shown by efforts at Google and Teradata to manage petabyte levels. Exabyte is generally considered as the current big data level as demonstrated by internet traffic rates. Zettabyte and yottabyte are emerging big data levels with hypothetical examples of storage capacity needs.

Table 3-4: Data Unit Sizes for Big Data

Data Unit	Big Data Example
Terabyte (TB)	Typical hard drive size on a personal computer in 2010
1,024 (1,000) GB	
Petabyte (PB)	Teradata Database 12 has a capacity of 50 petabytes of compressed data.
1,024 (1,000) TB	Google processes about 24 petabytes of data per day in 2011.
Exabtye (EB)	Global internet traffic was 21 exabytes in March 2010.
1,024 (1,000) PB	Estimate of global IP traffic in 2013 by Cisco is 667 exabytes
Zettabyte (ZB)	Cisco estimate of total volume of IP traffic in 2016 is 1.4 ZB.
1,024 (1,000) EB	
Yottabyte (YB)	High definition video of all human activity would be approximately 100 yottabytes. Estimate in 2014 of storage capacity of U.S. National Security data center capacity is 1.0 YB.
1,024 (1,000) ZB	

3.2 Processes for Data Warehouse Specialists

This section describes processes performed by data warehouse specialists. Data warehouse specialists participate in data governance sometimes serving on the data governance committee and other times consulting on activities managed by the data governance committee. Data warehouse specialists may perform tasks in the process of selecting and evaluating DBMSs especially for data warehouse features as well as supporting software for data integration. This section presents the details of both processes.

3.2.1 Data Governance Processes and Tools

Data governance overlaps somewhat with data planning sometimes done with developing an enterprise model for a data warehouse. Data governance emphasizes controls and accountability while data planning emphasizes organization-wide needs for tactical and strategic decision making. For example, enterprise data model development involves assessment of available data sources and related data quality but does not typically involve the controls and compliance representation in data governance models.

Microsoft and the Data Governance Institute have important proposals for data governance. The Microsoft approach provides some useful modeling tools so its approach is covered in this section. Otherwise, the approaches are reasonably similar. For more details about both approaches, you should see references listed in this section and at the end of the chapter.

The Microsoft Data Governance for Privacy, Confidentiality, and Compliance (DGPC) Framework contains components[3] for people, process, and technology as depicted in Figure 3.5. The people part of the framework involves a

3 Salido, J. and Voon, P. "A Guide to Data Governance for Privacy, Confidentiality, and Compliance (Part 2): People and Process," Microsoft Corporation, Whitepaper, January 2010.

pyramid with a small group of executive management providing strategic direction to a data governance organization. The data governance organization contains a diverse group of data stewards organized into steering committees and working groups. A data steward, typically a manager from a business area or subject area expert, is responsible for policies and standards applying to selected data. At the bottom of the pyramid, data consumers adhere to the policies and standards established by the data governance organization. Data consumers also report on violations of policies and standards for corrective action performed by data stewards and the data governance organization.

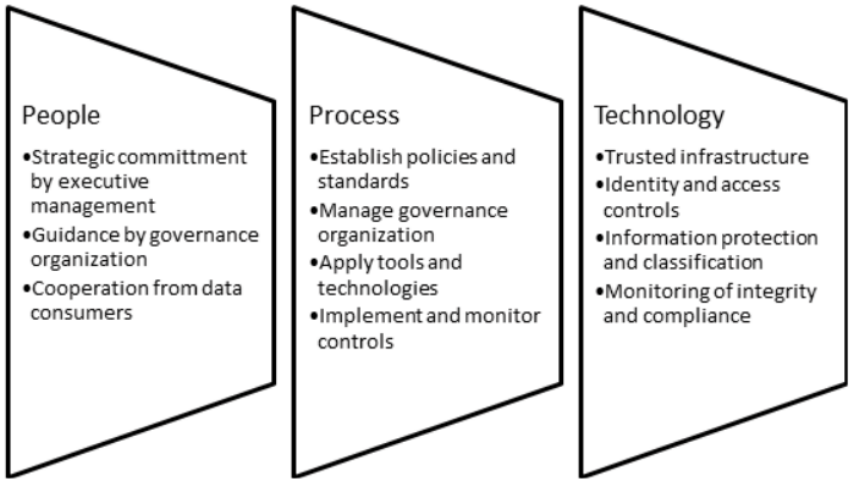


Figure 3.5: Components of the Microsoft DGPC Framework

(Adapted from Salido and Voon, 2010, Part 2)

Data Steward: an individual responsible for policies and standards applying to selected data. Typically managers from business areas or experts from subject areas serve as data stewards.

The DGPC Framework contains four core processes for managing the governance organization, requirements, strategies/policies, and controls as depicted in Figure 3.6. Managing the data governance organization involves processes for appointing members, defining roles and responsibilities, creating working groups, and reporting status and performance of data governance initiatives. Managing requirements involves translating business strategy into data quality and compliance requirements and collecting and integrating authority documents including regulations, standards, and policies. Managing strategies and policies involves processes to review, approve, publish, and implement authority documents. Managing the control environment involves processes for developing and monitoring controls and using tools to model information lifecycles and technology domains. The Microsoft information cycle is similar to the lifecycle shown in Figure 3.2. Table 3-5 lists functions of the four key technology domains: secure infrastructure, identity and access controls, information protection, and auditing and reporting.

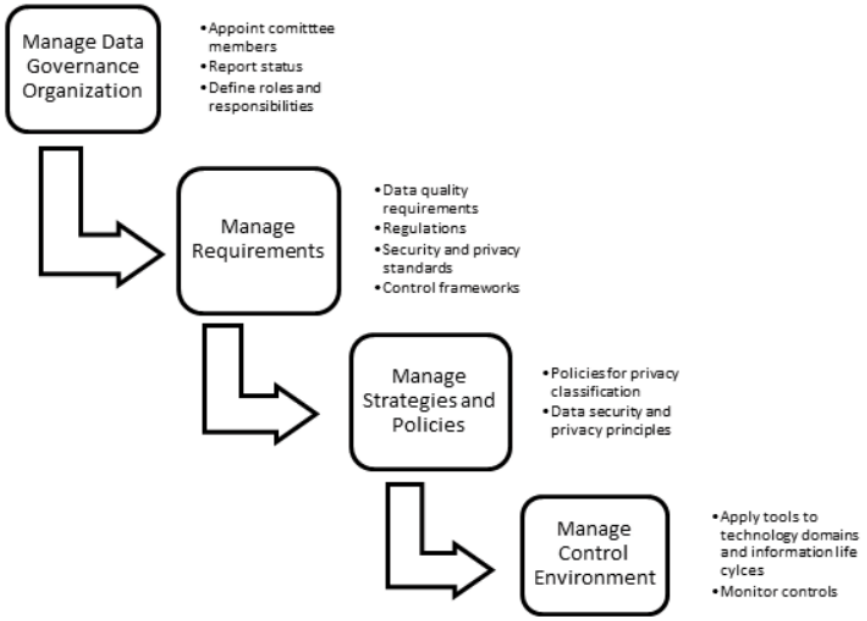


Figure 3.6: Core Processes in the Microsoft DGPC Framework

(Adapted from Adapted from Salido and Voon, 2010, Part 2)

Table 3-5: List of Functions in Microsoft Security Domains

Security Domain	Functions
Secure infrastructure	Stop malware and intrusions
	Monitor systems from evolving threats
Identity and access control	Safeguard sensitive data from unauthorized access or use
	Support controls for identity, access, and provisioning
Information protection	Protect sensitive data in databases, documents, messages, and records
	Safeguard data in motion
	Automate data classification for privacy
Auditing and reporting	Monitor integrity of systems and data
	Monitor compliance of data privacy and confidentiality with policies

Adapted from Adapted from Salido and Voon, 2010, Part 2

Controls, tools for managing risks to data assets, can be classified according to their timing (preventative, detective, and corrective) and automation level (manual, technology-aided, and automatic). Preventative controls are applied before an event occurs to ensure compliance. Typical preventative controls are segregation of duties and approval levels. Detective controls are applied after an event occurs to determine errors and irregularities. Exception reports, reconciliations, and audits are typical detective controls. Corrective controls are applied after detection of errors and noncompliance. Malware removal and backup restore are typical corrective controls. The level of automation depends on the type of control. Corrective controls are easiest to automate. Some human involvement is usually necessary for preventative and detective controls to perform procedures or review control results.

Control: a tool to manage risks to data assets. Controls can be classified according to their timing (preventative, detective, and corrective) and automation level (manual, technology-aided, and automatic).

The Risk-Gap Analysis Matrix[4] combines the information lifecycle and technology domains. The rows of the matrix are the data lifecycle stages while the columns are the technology domains as shown in Table 3-6. The last column is manual controls, not part of the technology domains. The cells show gaps in measures to protect data and manage risks for combinations of lifecycle activity and security domain. Gaps should be evaluated using compliance, data privacy, and confidentiality principles applicable to the combination of a lifecycle stage and technology domain.

Table 3-6: Template Risk-Gap Matrix in the Microsoft DGPC Framework

	Technology Domain				
Lifecycle Stage *Collect*	*Secure Infrastructure*	*Identity and Access Control*	*Information Protection*	*Auditing and Reporting*	*Manual Controls*
Update					
Process					
Delete					
Storage					
Transfer					

Adapted from Adapted from Salido and Voon, 2010, Part 3

Microsoft provides a risk-gap analysis process to support identification and resolution of gaps as depicted in Figure 3.7. In the first step, the context is established through defining the business purpose and listing the specific privacy, security, and compliance objectives. In the second step, threats are identified using a threat modeling tool such as the Microsoft Security Development Lifecycle tool. In the third step, risks for each cell of the risk-gap matrix are determined using the security and privacy principles. In the fourth step, risk mitigation techniques are chosen subject to cost-benefit analyses. In the fifth step, the effectiveness of mitigation techniques are evaluated using data collected from data consumers.

4 Salido, J. and Voon, P. "A Guide to Data Governance for Privacy, Confidentiality, and Compliance (Part 3): Managing Technological Risk," Microsoft Corporation, Whitepaper, March 2010.

Figure 3.7: Microsoft Risk Gap Analysis Process
(Adapted from Salido and Voon, 2010, Part 3)

To gauge an organization's relative progress in data governance, Microsoft provides the DGPC Capability Maturity Model (CMM)[5]. The CMM helps organizations determine their current status, target future status goals, and create action plans to reach maturity targets. The CMM provides a timeline showing stages of maturity, a development process to determine activities to reach future target maturity targets, and a table with details about capabilities in each core area. As depicted in Table 3-7, the stages of the CMM depend on the levels of training for personnel, development and integration of processes, and automation and integration of controls. The CMM development process involves understanding CMM capabilities, determining target levels, prioritizing development of new capabilities, and measuring progress. The main artifact of the CMM development process is a detailed table with main sections for people, process, and technology. Each area is evaluated on the four maturity levels with capabilities noted as current, planned, in progress, delivered, and adopted.

Table 3-7: Stages of the Microsoft Capability Maturity Model

Stage	People	Process	Technology
Basic	Lack of training and awareness	Few, immature processes	Lack of tools without integration
Standardized	Formal training	Established and communicated processes	Minimal tools for foundation goals
Rationalized	Formal training with compliance metrics	Process improvement	Increased usage of automated controls and some integration
Dynamic	Formal training with compliance metrics	Integrated compliance efforts	Full usage of automated and technology-aided controls with integration

Adapted from Adapted from Salido and Voon, 2010, Part 4

Data administrators and database administrators are usually involved in data governance. Data administrators participate through membership in the data governance office and consulting on activities managed by the data governance office. With their broad outlook on corporate data assets and interaction with various data stewards, data administrators are well-suited to manage a data governance office. Data administrators can also be heavily involved in setting

5 Salido, J. and Voon, P. "A Guide to Data Governance for Privacy, Confidentiality, and Compliance (Part 4): A Capability Maturity Model," Microsoft Corporation, Whitepaper, April 2010.

policies for data definitions, data quality, and controls. Database administrators usually serve a support role to help implement technology-aided and automated controls.

3.2.2 Selection and Evaluation of Database Management Systems

Selection and evaluation of a DBMS can be a very important task for an organization. DBMSs provide an important part of the computing infrastructure. As organizations strive to conduct electronic commerce over the Internet and extract value from operational databases, DBMSs play an even greater role. The selection and evaluation process is important because of the impacts of a poor choice. The immediate impacts may be slow database performance and loss of the purchase price. A poorly performing information system can cause lost sales and higher costs. The longer-term impacts are high switching costs. To switch DBMSs, an organization may need to convert data, recode software, and retrain employees. The switching costs can be much larger than the original purchase price.

The selection and evaluation process can also be performed for other major software products associated with data warehouses. Data integration software has grown in complexity and importance so it should be selected by a similar process. DBMSs have many features for data warehouse processing so data warehouse needs must be carefully considered in the selection of a DBMS.

Selection and Evaluation Process

The selection and evaluation process involves a detailed assessment of an organization's needs and features of candidate DBMSs. The goal of the process is to determine a small set of candidate systems that will be investigated in more detail. Because of the detailed nature of the process, a DBA performs most of the tasks. Therefore, a DBA needs a thorough knowledge of DBMSs to perform the process.

Figure 3.8 depicts the steps of the selection and evaluation process. In the first step, a DBA conducts a detailed analysis of the requirements. Because of the large number of requirements, it is helpful to group them. Table 3-8 lists major groupings of requirements while Table 3-9 shows some individual requirements in one group. Each individual requirement should be classified as essential, desirable, or optional to the requirement group. In some cases, several levels of requirements may be necessary. A DBA should be able to objectively measure individual requirements in the candidate systems.

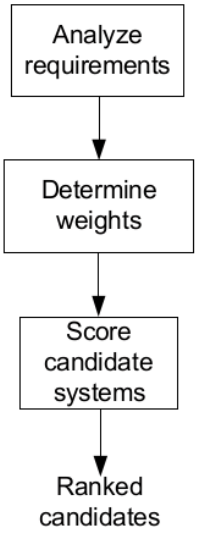


Figure 3.8: Overview of the Selection and Evaluation Process

Table 3-8: Some Major Requirement Groups

Category
Data definition (conceptual)
Nonprocedural retrieval
Data definition (internal)
Application development
Procedural language
Concurrency control
Recovery management
Parallel database processing
Distributed database support
Vendor support
Query optimization

Table 3-9: Some Detailed Requirements for the Conceptual Data Definition Category

Requirement (Importance)	Explanation
Entity integrity (essential)	Declaration and enforcement of primary keys
Candidate keys (desirable)	Declaration and enforcement of candidate keys
Referential integrity (essential)	Declaration and enforcement of referential integrity
Referenced rows (desirable)	Declaration and enforcement of rules for referenced rows
Standard data types (essential)	Support for whole numbers (several sizes), floating-point numbers (several sizes), fixed-point numbers, fixed-length strings, variable-length strings, and dates (date, time, and timestamp)
User-defined data types (desirable)	Support for new data types or a menu of optional data types
User interface (desirable)	Graphical user interface to manipulate dictionary tables
General assertions (optional)	Declaration and enforcement of multitable constraints
CHECK constraints (essential)	Declaration and enforcement of intra table constraints

After determining the groupings, the DBA should assign weights to the major requirement groups and score candidate systems. With more than a few major requirement groups, assigning consistent weights is very difficult. The DBA needs a tool to help assign consistent weights and to score candidate systems. The Analytic Hierarchy Process (AHP) provides a simple approach that achieves a reasonable level of consistency. The AHP has been used in a variety of management decision-making situations. As evidence of AHP's acceptance, a number of commercial software tools support the AHP.

Using the AHP, you assign weights to pairwise combinations of requirement groups using the nine-point scale in Table 3-10. For example, you should assign a weight that represents the importance of conceptual data definition as compared to nonprocedural retrieval. As shown in Table 3-11, the ranking of 5 in row 2, column 1 means that nonprocedural retrieval is significantly more important than conceptual data definition. For consistency, if entry $A_{ij} = x$, then $A_{ji} = 1/x$. Thus, the entry in row 1, column 2 is 1/5. The diagonal elements (comparing a requirement group to itself) should always be 1. Thus, it is necessary to complete only half the rankings in Table 3-11. The final row in the Table 3-11 shows column sums used to normalize weights and determine importance values.

Analytic Hierarchy Process: a decision theory technique to evaluate problems with multiple objectives. The process supports the selection and evaluation process by allowing a systematic assignment of weights to requirements and scores to features of candidate DBMSs.

Table 3-10: Interpretation of Rating Values for Pairwise Comparisons

Ranking Value of A_{ij}	Meaning
1	Requirements *i* and *j* are equally important.
3	Requirement *i* is slightly more important than requirement *j*.
5	Requirement *i* is significantly more important than requirement *j*.
7	Requirement *i* is very significantly more important than requirement *j*.
9	Requirement *i* is extremely more important than requirement *j*.

Table 3-11: Sample Weights for Some Requirement Groups

	Data Definition (Conceptual)	Nonprocedural Retrieval	Application Development	Concurrency Control
Data Definition (conceptual)	1	1/5 (0.20)	1/3 (0.33)	1/7 (0.14)
Nonprocedural Retrieval	5	1	3	1/3 (0.33)
Application Development	3	1/3 (0.33)	1	1/5 (0.20)
Concurrency Control	7	3	5	1
Column Sum	16	4.53	9.33	1.67

After assigning pairwise weights to the requirement groups, the weights are combined to determine an importance weight for each requirement group. The cell values are normalized by dividing each cell by its column sum as shown in Table 3-12. The final importance value for each requirement group is the average of the normalized weights in each row as shown in Table 3-13.

Table 3-12: Normalized Weights for Some Requirement Groups

	Data Definition (Conceptual)	Nonprocedural Retrieval	Application Development	Concurrency Control
Data Definition (conceptual)	0.06	0.04	0.04	0.08
Nonprocedural Retrieval	0.31	0.22	0.32	0.20
Application Development	0.19	0.07	0.11	0.12
Concurrency Control	0.44	0.66	0.54	0.60

Table 3-13: Importance Values for Some Requirement Groups

Requirement Group	Importance
Data Definition (conceptual)	0.06
Nonprocedural Retrieval	0.26
Application Development	0.12
Concurrency Control	0.56

Importance weights must be computed for each subcategory of requirement groups in the same manner as for requirement groups. For each subcategory, pairwise weights are assigned before normalizing the weights and computing final importance values.

After computing importance values for the requirements, candidate DBMSs are assigned scores. Scoring candidate DBMSs can be complex because of the number of individual requirements and the need to combine individual requirements into an overall score for the requirement group. As the first part of the scoring process, a DBA should carefully

investigate the features of each candidate DBMS.

Many approaches have been proposed to combine individual feature scores into an overall score for the requirement group. The Analytic Hierarchy Process supports pairwise comparisons among candidate DBMSs using the rating values in Table 3-18. The interpretations change slightly to reflect comparisons among candidate DBMSs rather than the importance of requirement groups. For example, a value 3 should be assigned if DBMS *i* is slightly better than DBMS *j*. For each requirement subcategory, a comparison matrix should be created to compare the candidate DBMSs. Scores for each DBMS are computed by normalizing the weights and computing the row averages as for requirement groups.

After scoring the candidate DBMSs for each requirement group, the final scores are computed by combining the requirement group scores with the importance of requirement groups. For details about computing the final scores, you should consult the references at the end of the chapter about the Analytic Hierarchy Process.

Final Selection Process

After the selection and evaluation process completes, the top two or three candidate DBMSs should be evaluated in more detail. Benchmarks can be used to provide a more detailed evaluation of candidate DBMSs. A benchmark is a workload to evaluate the performance of a system or product. A good benchmark should be relevant, portable, scalable, and understandable. Because developing good benchmarks requires significant expertise, most organizations should not attempt to develop a benchmark. Fortunately, the Transaction Processing Council (TPC) has developed a number of standard, domain-specific benchmarks as summarized in Table 3-14. Each benchmark was developed over an extended time period with input from a diverse group of contributors.

Benchmark: a workload to evaluate the performance of a system or product. A good benchmark should be relevant, portable, scalable, and understandable.

Table 3-14: Summary of Currently Supported[6] TPC Benchmarks

Benchmark	Description	Performance Measures
TPC-C	Online order entry benchmark	Transactions per minute, price per transactions per minute
TPC-H	Decision support for ad hoc queries	Composite queries per hour, price per composite queries per hour
TPC-E	Transaction workload of a brokerage firm	Transactions per second, price per transactions per second
TPC-DS	Decision support benchmark for a retail product supplier; uses both queries and data integration processing	Effective query throughput, price performance, system availability

The currently supported benchmarks are extended with specifications for pricing, energy efficiency, and virtual measurement. The pricing specification was developed to reduce confusion caused by special pricing policies offered by vendors especially benchmark pricing. Vendor prices should be generally available, published prices for commercial products to evaluate price/performance for a three year period. Each vendor must disclose the purchase price of associated hardware, software licensing costs, and maintenance contracts. The pricing specification stipulates auditing requirements for the full disclosure agreement about prices provided in a benchmark result.

In response to the growing importance of energy efficiency in information technology selection, the TPC developed the TPC-Energy specification. Customers have increasingly demanded both price/performance and energy/performance results for information technology purchasing decisions. The TPC-Energy specification provides standards for energy metrics and a software tool to support energy measurement and reporting by benchmark vendors. The energy metric standard stipulates components of the system under test and aspects of the physical environment (such as temperature, humidity, and altitude) for execution of a benchmark. The Energy Metric System supports a Web interface for power system instrumentation, power and temperature logging, and report generation.

The virtual measurement specification supports benchmark execution and reporting on virtual databases. Organizations increasingly utilize virtualization environments leading to demands for benchmark measurements for virtual databases.

6 In addition to the currently supported benchmarks, the TPC has six obsolete benchmarks.

A DBA can use the TPC results for each benchmark to obtain reasonable estimates about the performance of a particular DBMS in a specific hardware/software environment. The TPC performance results involve total system performance, not just DBMS performance so that results are not inflated when a customer uses a DBMS in a specific hardware/software environment. To facilitate price performance trade-offs, the TPC publishes the performance measure along with price/performance for each benchmark. The price covers all cost dimensions of an entire system environment including workstations, communications equipment, system software, computer system or host, backup storage, and three years' maintenance cost. The TPC audits the benchmark results prior to publication to ensure that vendors have not manipulated the results.

To augment the published TPC results, an organization may want to evaluate a DBMS on a trial basis. Customized benchmarks can be created to gauge the efficiency of a DBMS for its intended usage. In addition, the user interface and the application development capabilities can be evaluated by building small applications.

The final phase of the selection process may involve nontechnical considerations performed by data administrators along with senior management and legal staff. Assessment of each vendor's future prospects is important because information systems can have a long life. If the underlying DBMS does not advance with the industry, it may not support future initiatives and upgrades to the information systems that use it. Because of the high fixed and variable costs (maintenance fees) of a DBMS, negotiation is often a critical element of the final selection process. The final contract terms along with one or two key advantages often make the difference in the final selection.

Open source DBMS software is a recent development that complicates the selection and evaluation process. Open source DBMS software has uncertainty in licensing and future prospects but obvious purchase price advantages over commercial DBMS software. With open source software, the lack of profit incentives may hinder product updates and lead to software license changes to obtain product updates. For example, MySQL, the most popular open source DBMS, changed its licensing so that commercial users will typically pay licensing fees. Despite these uncertainties, many organizations utilize open source DBMS software especially for non-mission-critical systems.

Closing Thoughts

This chapter has described the responsibilities and processes used by data specialists to manage data warehouse and support management decision making. Many organizations provide two roles for managing information resources. Data administrators perform broad planning and policy setting, while database administrators perform detailed oversight of individual databases and DBMSs. To provide a context to understand the responsibilities of data specialists, this chapter described programs for managing organizational information and knowledge as well as governance of information resources. In many organizations, data specialists participate in these programs in an environment dominated by challenges and opportunities from exploding data growth.

Database specialists need to understand two important processes to manage data warehouse technology. Data administrators often serve in key roles in data governance, a process to control risks to data assets and improve compliance with privacy and confidentiality policies. Database administrators may support data governance by helping to implement technology-aided controls. This chapter described the Microsoft Framework for Data Governance, Privacy, Confidentiality, and Compliance, a prominent approach involving people, processes, and technology components to support data governance activities. Both data administrators and database administrators participate in the selection and evaluation of DBMSs and related data warehouse software packages. Database administrators perform the detailed tasks while data administrators often make final selection decisions based on the detailed recommendation and negotiation with vendors. This chapter described the steps of the selection and evaluation process and the tasks performed by data warehouse specialists in the process.

Review Concepts

- Information resource management: management philosophy to control information resources and apply information technology to support management decision-making

- Data governance involving the application of decision-making and authority for data-related issues
- Database administrator: support position for managing individual databases and DBMSs
- Data administrator: management position with planning and policy responsibilities for information technology
- Challenges and opportunities dealing with big data, the phenomenon of exploding growth of data volumes, velocity, and variety
- Establishment of an organization to develop, implement, and monitor policies and standards for data governance
- Developing controls to detect, prevent, and correct violations of integrity, security, and privacy policies
- Selection and evaluation process for matching organization needs to DBMS features
- Using a tool such as the Analytic Hierarchy Process for consistently assigning importance weights and scoring candidate DBMSs and related data warehouse products
- Using TPC benchmark results to gauge the performance of DBMSs
- Responsibilities of database specialists for managing transaction processing, data warehouses, distributed environments, and object DBMSs

Questions

1. Why is it difficult to use operational databases for management decision making?
2. How must operational databases be transformed for management decision making?
3. What are the phases of the information life cycle?
4. What does it mean to integrate information life cycles?
5. What data quality dimension is important for management decision making but not for operational decision making?
6. How does knowledge management differ from information resource management?
7. What are the three pillars of knowledge management?
8. What kind of position is the data administrator?
9. What kind of position is the database administrator?
10. Which position (data administrator versus database administrator) takes a broader view of information resources?
11. What kinds of specialization are possible in large organizations for data administrators and database administrators?
12. Why is the selection and evaluation process important for DBMSs?
13. What are some difficulties in the selection and evaluation process for a complex product like a DBMS?
14. What are the steps in the selection and evaluation process?
15. How is the Analytic Hierarchy Process used in the selection and evaluation process?
16. What responsibilities does the database administrator have in the selection and evaluation process?
17. What responsibilities does the data administrator have in the selection and evaluation process?

18. What are responsibilities of database administrators for managing data warehouses?
19. What are the characteristics of a good benchmark?
20. Why does the Transaction Processing Council publish total system performance measures rather than component measures?
21. Why does the Transaction Processing Council publish price/performance results?
22. How does the Transaction Processing Council ensure that benchmark results are relevant and reliable?
23. What is the TPC pricing specification?
24. What is the TPC energy efficiency specification?
25. What is a data steward? What role do data stewards play in data governance?
26. Briefly describe the types of controls and discuss the role of controls in the data governance process.
27. How is the Risk-Gap Matrix used in Microsoft's data governance framework?
28. How is the Capability Maturity Model used in Microsoft's data governance framework?
29. Provide two definitions for big data.
30. List the major sources of big data.
31. Briefly explain the ways that big data creates business opportunities.
32. Provide examples of the units of big data.

Problems

Due to the nature of this chapter, the problems are more open-ended than other chapters. More detailed problems appear at the end of Chapters 1 and 2.

1. Prepare a short presentation (6 to 12 slides) about the TPC-C benchmark. You should provide details about its history, database design, application details, and recent results.
2. Prepare a short presentation (6 to 12 slides) about the TPC-H benchmark. You should provide details about its history, database design, application details, and recent results.
3. Prepare a short presentation (6 to 12 slides) about the TPC-E benchmark. You should provide details about its history, database design, application details, and recent results.
4. Prepare a short presentation (6 to 12 slides) about the TPC-App benchmark. You should provide details about its history, database design, application details, and recent results. Note that the TPC-App benchmark has been recently obsoleted by the TPC.
5. Compare and contrast the software licenses for MySQL and another open source DBMS product.
6. Develop a list of detailed requirements for nonprocedural retrieval. You should use Table 3-9 as a guideline.
7. Provide importance weights for your list of detailed requirements from problem 6 using the AHP criteria in Table 3-11.
8. Normalize the weights and compute the importance values for your detailed requirements using the importance weights from problem 7.
9. Investigate the data governance practices of a profit or government organization. You will need to obtain permission to interview individuals with knowledge of data governance practices at your chosen organization. You

should investigate the structure of the data governance organization, processes used by the data governance organization, and projects in which the data governance organization has attempted. You should prepare a slide show presentation to communicate your findings. You may need to disguise the organization and individuals interviewed for privacy considerations.

References for Further Study

Mullins (2012) provides a recent comprehensive reference about database administration as well as a website (www.craigsmullins.com) with many articles about database administration. The Database Trends and Applications website (www.dbta.com) contains current details about products and industry developments. Microsoft (www.microsoft.com/privacy/datagovernance.aspx) and the Data Governance Institute (www.datagovernance.com) provide whitepapers, case studies and other resources about data governance. Gartner (www.gartner.com) and the McKinsey Global Institute (www.mckinsey.com/insights/mgi.aspx) provide analysis of big data challenges and opportunities. For more details about the Analytic Hierarchy Process mentioned in Section 14.3.2, you should consult Saaty (1988) and Zahedi (1986). Some excellent online AHP tutorials can be found by searching the Web on "Analytical Hierarchy Process Tutorials". Su et al. (1987) describe the Logic Scoring of Preferences, an alternative approach to DBMS selection. The Transaction Processing Council (www.tpc.org) provides an invaluable resource about domain-specific benchmarks for DBMSs.