8 Business Intelligence

Business intelligence has become a buzzword in recent years. The database tools found under the heading of *business intelligence* include data warehousing, online analytical processing (OLAP), and data mining. The functionalities of these tools are complementary and interrelated. *Data warehousing* provides for the efficient storage, maintenance, and retrieval of historical data. *OLAP* is a service that provides quick answers to *ad hoc* queries against the data warehouse. *Data mining* algorithms find patterns in the data and report models back to the user. All three tools are related to the way data in a data warehouse are logically organized, and performance is highly sensitive to the database design techniques used [Barquin and Edelstein, 1997]. The encompassing goal for business intelligence technologies is to provide useful information for decision support.

Each of the major DBMS vendors is marketing the tools for data warehousing, OLAP, and data mining as business intelligence. This chapter covers each of these technologies in turn. We take a close look at the requirements for a data warehouse; its basic components and principles of operation; the critical issues in its design; and the important logical database design elements in its environment. We then investigate the basic elements of OLAP and data mining as special query techniques applied to data warehousing. We cover data warehousing in Section 8.1, OLAP in Section 8.2, and data mining in Section 8.3.

8.1 Data Warehousing

A data warehouse is a large repository of historical data that can be integrated for decision support. The use of a data warehouse is markedly different from the use of operational systems. Operational systems contain the data required for the day-to-day operations of an organization. This operational data tends to change quickly and constantly. The table sizes in operational systems are kept manageably small by periodically purging old data. The data warehouse, by contrast, periodically receives historical data in batches, and grows over time. The vast size of data warehouses can run to hundreds of gigabytes, or even terabytes. The problem that drives data warehouse design is the need for quick results to queries posed against huge amounts of data. The contrasting aspects of data warehouses and operational systems result in a distinctive design approach for data warehousing.

8.1.1 Overview of Data Warehousing

A data warehouse contains a collection of tools for decision support associated with very large historical databases, which enables the end user to make quick and sound decisions. Data warehousing grew out of the technology for decision support systems (DSS) and executive information systems (EIS). DSSs are used to analyze data from commonly available databases with multiple sources, and to create reports. The report data is not time critical in the sense that a real-time system is, but it must be timely for decision making. EISs are like DSSs, but more powerful, easier to use, and more business specific. EISs were designed to provide an alternative to the classical online transaction processing (OLTP) systems common to most commercially available database systems. OLTP systems are often used to create common applications, including those with mission-critical deadlines or response times. Table 8.1 summarizes the basic differences between OLTP and data warehouse systems.

The basic architecture for a data warehouse environment is shown in Figure 8.1. The diagram shows that the data warehouse is stocked by a variety of source databases from possibly different geographical locations. Each source database serves its own applications, and the data warehouse serves a DSS/EIS with its informational requests. Each feeder system database must be reconciled with the data warehouse data model; this is accomplished during the process of extracting the required data from the feeder database system, transforming the data

Table 8.1 Comparison between OLTP and Data Warehouse Databases

OLTP	***Data Warehouse***
Transaction oriented	Business process oriented
Thousands of users	Few users (typically under 100)
Generally small (MB up to several GB)	Large (from hundreds of GB to several TB)
Current data	Historical data
Normalized data (many tables, few columns per table)	Denormalized data (few tables, many columns per table)
Continuous updates	Batch updates*
Simple to complex queries	Usually very complex queries

* There is currently a push in the industry towards "active warehousing," in which the warehouse receives data in continuous updates. See Section 8.2.5 for further discussion.

from the feeder system to the data warehouse, and loading the data into the data warehouse [Cataldo, 1997].

Core Requirements for Data Warehousing

Let us now take a look at the core requirements and principles that guide the design of data warehouses (DWs) [Simon, 1995; Barquin and Edelstein, 1997; Chaudhuri and Dayal, 1997; Gray and Watson, 1998]:

1. DWs are organized around subject areas. Subject areas are analogous to the concept of functional areas, such as sales, project management, or employees, as discussed in the context of ER diagram clustering in Section 4.5. Each subject area has its own conceptual schema and can be represented using one or more entities in the ER data model or by one or more object classes in the object-oriented data model. Subject areas are typically independent of individual transactions involving data creation or manipulation. Metadata repositories are needed to describe source databases, DW objects, and ways of transforming data from the sources to the DW.
2. DWs should have some integration capability. A common data representation should be designed so that all the different individual representations can be mapped to it. This is particularly

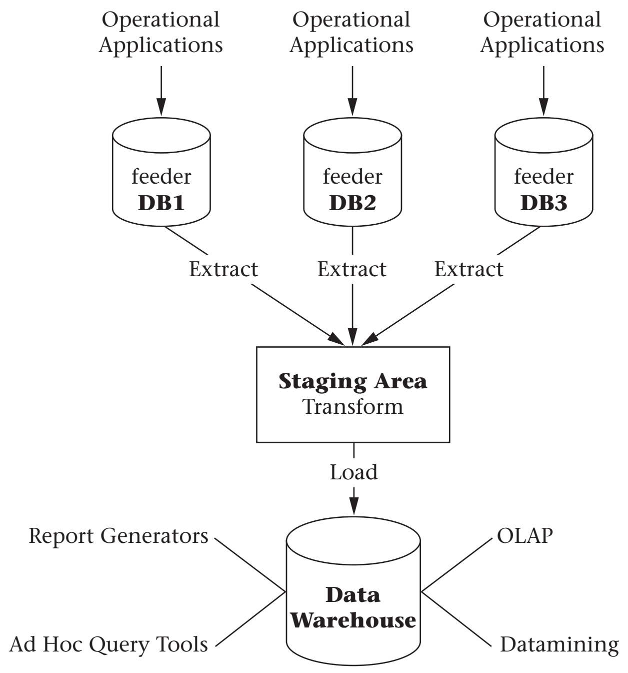


Figure 8.1 Basic data warehouse architecture

useful if the warehouse is implemented as a multidatabase or federated database.

3. The data is considered to be nonvolatile and should be mass loaded. Data extraction from current databases to the DW requires that a decision should be made whether to extract the data using standard relational database (RDB) techniques at the row or column level or specialized techniques for mass extraction. Data cleaning tools are required to maintain data quality—for example, to detect missing data, inconsistent data, homonyms, synonyms, and data with different units. Data migration, data scrubbing, and data auditing tools handle specialized problems in data cleaning and transformation. Such tools are similar to those used for conventional relational database schema (view) integration. Load utilities take cleaned data and load it into the DW, using batch processing techniques. Refresh techniques propagate updates on the source data to base data and derived data in the DW. The decision of when and how to refresh is made by the DW

administrator and depends on user needs (e.g., OLAP needs) and existing traffic to the DW.

4. Data tends to exist at multiple levels of granularity. Most important, the data tends to be of a historical nature, with potentially high time variance. In general, however, granularity can vary according to many different dimensions, not only by time frame but also by geographic region, type of product manufactured or sold, type of store, and so on. The sheer size of the databases is a major problem in the design and implementation of DWs, especially for certain queries, updates, and sequential backups. This necessitates a critical decision between using a relational database (RDB) or a multidimensional database (MDD) for the implementation of a DW.
5. The DW should be flexible enough to meet changing requirements rapidly. Data definitions (schemas) must be broad enough to anticipate the addition of new types of data. For rapidly changing data retrieval requirements, the types of data and levels of granularity actually implemented must be chosen carefully.
6. The DW should have a capability for rewriting history, that is, allowing for "what-if" analysis. The DW should allow the administrator to update historical data temporarily for the purpose of "what-if" analysis. Once the analysis is completed, the data must be correctly rolled back. This condition assumes that the data are at the proper level of granularity in the first place.
7. A usable DW user interface should be selected. The leading choices today are SQL, multidimensional views of relational data, or a special-purpose user interface. The user interface language must have tools for retrieving, formatting, and analyzing data.
8. Data should be either centralized or distributed physically. The DW should have the capability to handle distributed data over a network. This requirement will become more critical as the use of DWs grows and the sources of data expand.

The Life Cycle of Data Warehouses

Entire books have been written about select portions of the data warehouse life cycle. Our purpose in this section is to present some of the basics and give the flavor of data warehousing. We strongly encourage those who wish to pursue data warehousing to continue learning through other books dedicated to data warehousing. Kimball and Ross

[1998, 2002] have a series of excellent books covering the details of data warehousing activities.

Figure 8.2 outlines the activities of the data warehouse life cycle, based heavily on Kimball and Ross's Figure 16.1 [2002]. The life cycle begins with a dialog to determine the project plan and the business requirements. When the plan and the requirements are aligned, design and implementation can proceed. The process forks into three threads that follow independent timelines, meeting up before deployment (see Figure 8.2). Platform issues are covered in one thread, including technical architectural design, followed by product selection and installation. Data issues are covered in a second thread, including dimensional modeling and then physical design, followed by data staging design and development. The special analytical needs of the users are pursued in the third thread, including analytic application specification followed by analytic application development. These three threads join before deployment. Deployment is followed by maintenance and growth, and changes in the requirements must be detected. If adjustments are needed, the cycle repeats. If the system becomes defunct, then the life cycle terminates.

The remainder of our data warehouse section focuses on the dimensional modeling activity. More comprehensive material can be found in Kimball and Ross [1998, 2002] and Kimball and Caserta [2004].

8.1.2 Logical Design

We discuss the logical design of data warehouses in this section; the physical design issues are covered in volume two. The logical design of data warehouses is defined by the dimensional data modeling approach. We cover the schema types typically encountered in dimensional modeling, including the star schema and the snowflake schema. We outline the dimensional design process, adhering to the methodology described by Kimball and Ross [2002]. Then we walk through an example, covering some of the crucial concepts of dimensional data modeling.

Dimensional Data Modeling

The dimensional modeling approach is quite different from the normalization approach typically followed when designing a database for daily operations. The context of data warehousing compels a different approach to meeting the needs of the user. The need for dimensional modeling will be

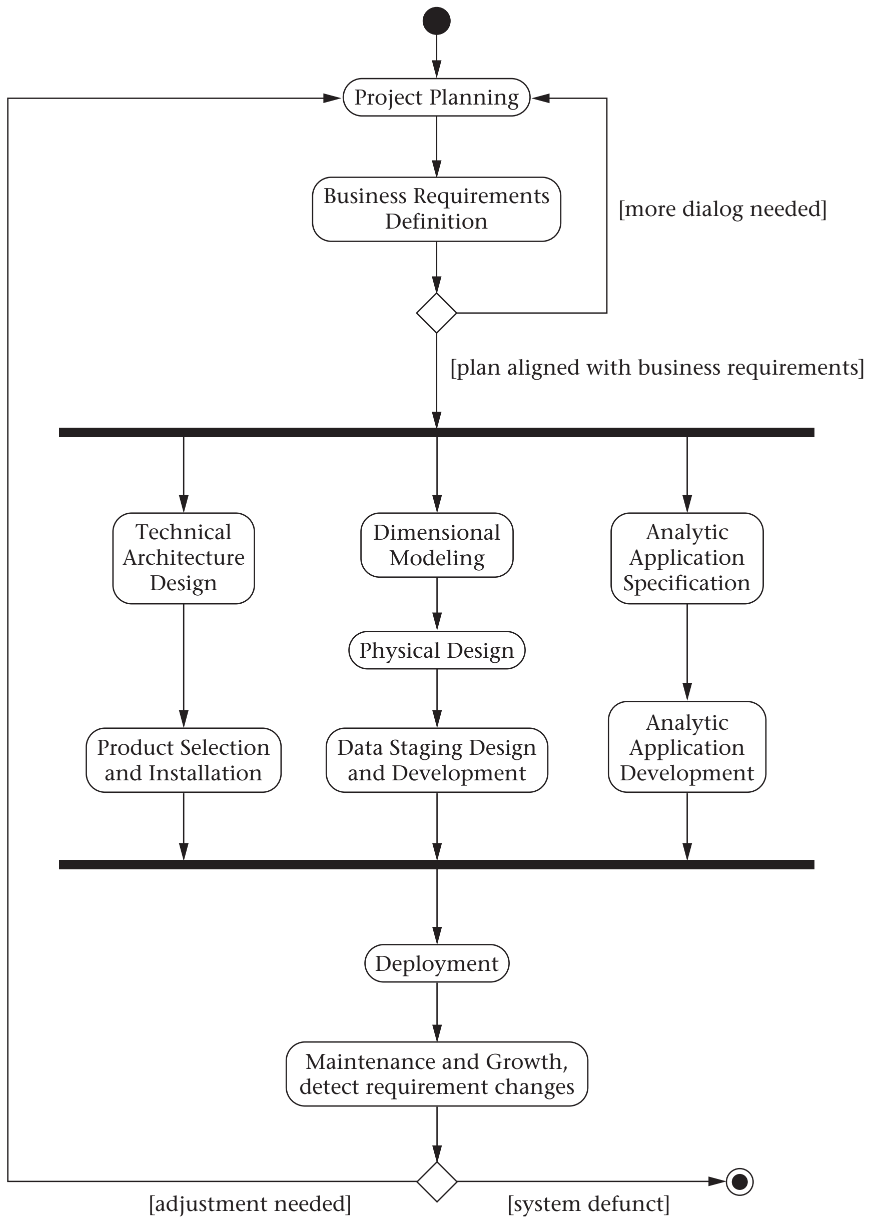

Project Planning
Business Requirements Definition
[more dialog needed]
[plan aligned with business requirements]
Technical Architecture Design
Dimensional Modeling
Analytic Application Specification
Physical Design
Product Selection and Installation
Data Staging Design and Development
Analytic Application Development
Deployment
Maintenance and Growth, detect requirement changes
[adjustment needed]
[system defunct]

Figure 8.2 Data warehouse life cycle (based heavily on Kimball and Ross [2002], Figure 16.1)

discussed further as we proceed. If you haven't been exposed to data warehousing before, be prepared for some new paradigms.

The Star Schema

Data warehouses are commonly organized with one large central *fact table*, and many smaller *dimension tables*. This configuration is termed a *star schema*; an example is shown in Figure 8.3. The fact table is composed of two types of attributes: *dimension attributes* and *measures*. The dimension attributes in Figure 8.3 are CustID, ShipDateID, BindID, and JobId. Most dimension attributes have foreign key/primary key relationships with dimension tables. The dimension tables in Figure 8.3 are Customer, Ship Calendar, and Bind Style. Occasionally, a dimension attribute exists without a related dimension table. Kimball and Ross refer to these as *degenerate dimensions*. The JobId attribute in Figure 8.3 is a degenerate dimension (more on this shortly). We indicate the dimension attributes that act as foreign keys using the stereotype «fk». The primary keys of the dimension tables are indicated with the stereotype «pk». Any degenerate dimensions in the fact table are indicated with the stereotype «dd». The fact table also contains measures, which contain values to be aggregated when queries group rows together. The measures in Figure 8.3 are Cost and Sell.

Queries against the star schema typically use attributes in the dimension tables to select the pertinent rows from the fact table. For example, the user may want to see cost and sell for all jobs where the Ship Month

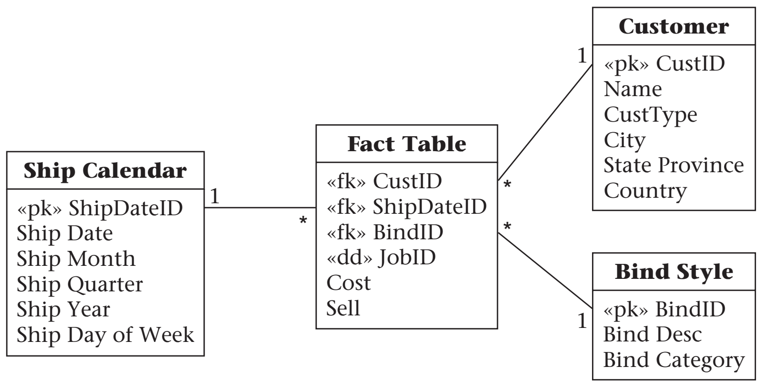


Figure 8.3 Example of a star schema for a data warehouse

is January 2005. The dimension table attributes are also typically used to group the rows in useful ways when exploring summary information. For example, the user may wish to see the total cost and sell for each Ship Month in the Ship Year 2005. Notice that dimension tables can allow different *levels* of detail the user can examine. For example, the Figure 8.3 schema allows the fact table rows to be grouped by Ship Date, Month, Quarter or Year. These dimension levels form a *hierachy.* There is also a second hierarchy in the Ship Calendar dimension that allows the user to group fact table rows by the day of the week. The user can move up or down a hierarchy when exploring the data. Moving down a hierarchy to examine more detailed data is a *drill-down* operation. Moving up a hierarchy to summarize details is a *roll-up* operation.

Together, the dimension attributes compose a candidate key of the fact table. The level of detail defined by the dimension attributes is the *granularity* of the fact table. When designing a fact table, the granularity should be the most detailed level available that any user would wish to examine. This requirement sometimes means that a degenerate dimension, such as JobId in Figure 8.3, must be included. The JobId in this star schema is not used to select or group rows, so there is no related dimension table. The purpose of the JobId attribute is to distinguish rows at the correct level of granularity. Without the JobId attribute, the fact table would group together similar jobs, prohibiting the user from examining the cost and sell values of individual jobs.

Normalization is not the guiding principle in data warehouse design. The purpose of data warehousing is to provide quick answers to queries against a large set of historical data. Star schema organization facilitates quick response to queries in the context of the data warehouse. The core detailed data are centralized in the fact table. Dimensional information and hierarchies are kept in dimension tables, a single join away from the fact table. The hierarchical levels of data contained in the dimension tables of Figure 8.3 violate 3NF, but these violations to the principles of normalization are justified. The normalization process would break each dimension table in Figure 8.3 into multiple tables. The resulting normalized schema would require more join processing for most queries. The dimension tables are small in comparison to the fact table, and typically slow changing. The bulk of operations in the data warehouse are read operations. The benefits of normalization are low when most operations are read only. The benefits of minimizing join operations overwhelm the benefits of normalization in the context of data warehousing. The marked differences between the data warehouse environment and the operational system environment lead to distinct design approaches. Dimensional modeling is the guiding principle in data warehouse design.

Snowflake Schema

The data warehouse literature often refers to a variation of the star schema known as the *snowflake schema*. Normalizing the dimension tables in a star schema leads to a snowflake schema. Figure 8.4 shows the snowflake schema analogous to the star schema of Figure 8.3. Notice that each hierarchical level becomes its own table. The snowflake schema is generally losing favor. Kimball and Ross strongly prefer the star schema, due to its speed and simplicity. Not only does the star schema yield quicker query response, it is also easier for the user to understand when building queries. We include the snowflake schema here for completeness.

Dimensional Design Process

We adhere to the four-step dimensional design process promoted by Kimball and Ross. Figure 8.5 outlines the activities in the four-step process.

Dimensional Modeling Example

Congratulations, you are now the owner of the ACME Data Mart Company! Your company builds data warehouses. You consult with other companies, design and deploy data warehouses to meet their needs, and support them in their efforts.

Your first customer is XYZ Widget, Inc. XYZ Widget is a manufacturing company with information systems in place. These are operational systems that track the current and recent state of the various business processes. Older records that are no longer needed for operating the plant are purged. This keeps the operational systems running efficiently.

XYZ Widget is now ten years old, and growing fast. The management realizes that information is valuable. The CIO has been saving data before they are purged from the operational system. There are tens of millions of historical records, but there is no easy way to access the data in a meaningful way. ACME Data Mart has been called in to design and build a DSS to access the historical data.

Discussions with XYZ Widget commence. There are many questions they want to have answered by analyzing the historical data. You begin by making a list of what XYZ wants to know.

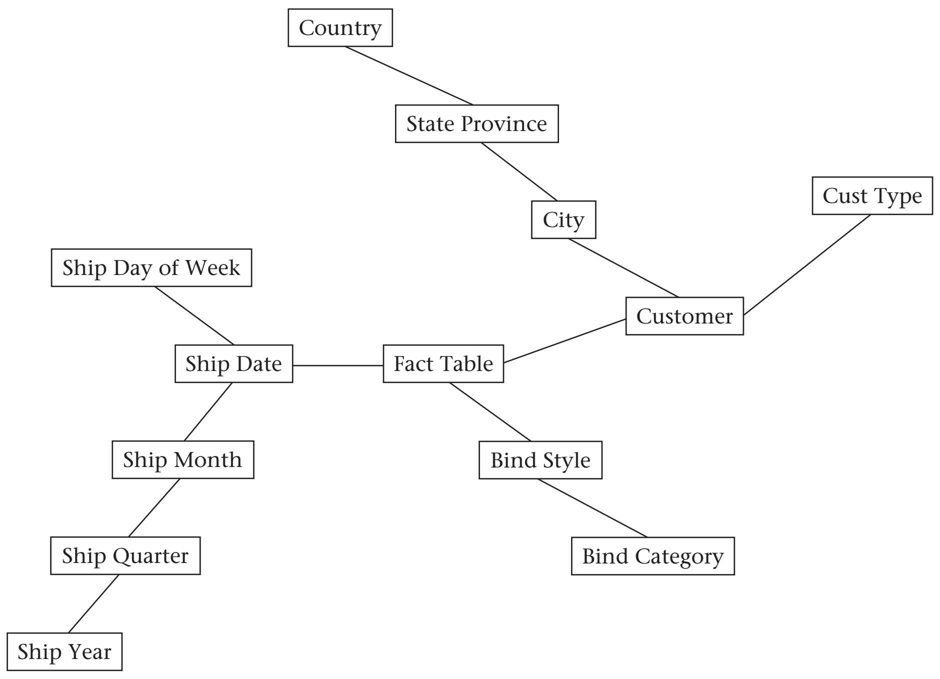


Figure 8.4 Example of a snowflake schema for a data warehouse

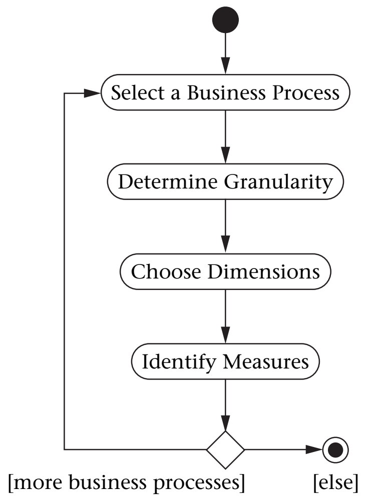


Figure 8.5 Four step dimensional design process [Kimball and Ross, 2002]

XYZ Widget Company Wish List

1. What are the trends of our various products in terms of sales dollars, unit volume, and profit margin?
2. For those products that are not profitable, can we drill down and determine why they are not profitable?
3. How accurately do our estimated costs match our actual costs?
4. When we change our estimating calculations, how are sales and profitability affected?
5. What are the trends in the percentage of jobs that ship on time?
6. What are the trends in productivity by department, for each machine, and for each employee?
7. What are the trends in meeting the scheduled dates for each department, and for each machine?
8. How effective was the upgrade on machine 123?
9. Which customers bring the most profitable jobs?
10. How do our promotional bulk discounts affect sales and profitability?

Looking over the wish list, you begin picking out the business processes involved. The following list is sufficient to satisfy the items on the wish list.

Business Processes

1. Estimating
2. Scheduling
3. Productivity Tracking
4. Job Costing

These four business processes are interlinked in the XYZ Widget Company. Let's briefly walk through the business processes and the organization of information in the operational systems, so we have an idea what information is available for analysis. For each business process, we'll design a star schema for storing the data.

The estimating process begins by entering widget specifications. The type of widget determines which machines are used to manufacture the widget. The estimating software then calculates estimated time on each machine used to produce that particular type of widget. Each machine is modeled with a standard setup time and running speed. If a particular type of widget is difficult to process on a particular machine, the times are adjusted accordingly. Each machine has an hourly rate. The estimated time is multiplied by the rate to give labor cost. Each estimate stores widget specifications, a breakdown of the manufacturing costs, the markup and discount applied (if any), and the price. The quote is sent to the customer. If the customer accepts the quote, then the quote is associated with a job number, the specifications are printed as a job ticket, and the job ticket moves to scheduling.

We need to determine the grain before designing a schema for the estimating data mart. The grain should be at the most detailed level, giving the greatest flexibility for drill-down operations when users are exploring the data. The most granular level in the estimating process is the estimating detail. Each estimating detail record specifies information for an individual cost center for a given estimate. This is the finest granularity of estimating data in the operational system, and this level of detail is also potentially valuable for the data warehouse users.

The next design step is to determine the dimensions. Looking at the estimating detail, we see that the associated attributes are the job specifications, the estimate number and date, the job number and win date if the estimate becomes a job, the customer, the promotion, the cost center, the widget quantity, estimated hours, hourly rate, estimated cost, markup, discount, and price. Dimensions are those attributes that the users want to group by when exploring the data. The users are interested in grouping by the various job specifications and by the cost center. The users also need to be able to group by date ranges. The estimate date and the win date are both of interest. Grouping by customer and promotion are also of interest to the users. These become the dimensions of the star schema for the estimating process.

Next, we identify the measures. Measures are the columns that contain values to be aggregated when rows are grouped together. The measures in the estimating process are estimated hours, hourly rate, estimated cost, markup, discount, and price.

The star schema resulting from the analysis of the estimating process is shown in Figure 8.6. There are five widget qualities of interest: shape, color, texture, density, and size. For example, a given widget might be a medium round red fuzzy fluffy widget. The estimate and job numbers are included as degenerate dimensions. The rest of the dimensions and measures are as outlined in the previous two paragraphs.

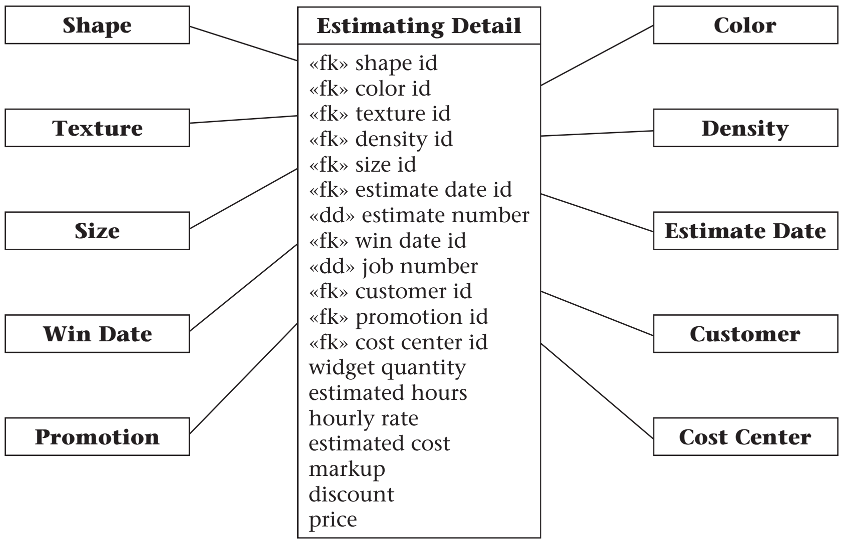


Figure 8.6 Star schema for estimating process

Dimension values are categorical in nature. For example, a given widget might have a density of fluffy or heavy. The values for the size dimension include small, medium, and large. Measures tend to be numeric, since they are typically aggregated using functions such as sum or average.

The dimension tables should include any hierarchies that may be useful for analysis. For example, widgets are offered in many colors. The colors fall into categories by hue (e.g., pink, blue) and intensity (e.g., pastel, hot). Some even glow in the dark! The user may wish to examine all the pastel widgets as a group, or compare pink versus blue widgets. Including these attributes in the dimension table as shown in Figure 8.7 can accommodate this need.

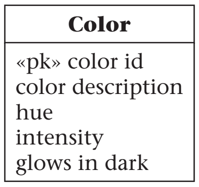


Figure 8.7 Color dimension showing attributes

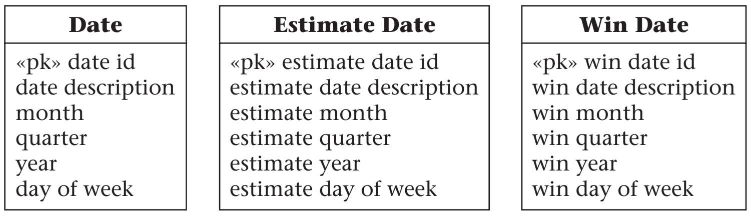


Figure 8.8 Date dimensions showing attributes

Dates can also form hierarchies. For example, the user may wish to group by month, quarter, year or the day of the week. Date dimensions are very common. The estimating process has two date dimensions: the estimate date and the win date. Typically, the date dimensions have analogous attributes. There is an advantage in standardizing the date dimensions across the company. Kimball and Ross [2002] recommend establishing a single standard date dimension, and then creating views of the date dimension for use in multiple dimensions. The use of views provides for standardization, while at the same time allowing the attributes to be named with aliases for intuitive use when multiple date dimensions are present. Figure 8.8 illustrates this concept with a date dimension and two views named Estimate Date and Win Date.

Let's move on to the scheduling process. Scheduling uses the times calculated by the estimating process to plan the workload on each required machine. Target dates are assigned to each manufacturing step. The job ticket moves into production after the scheduling process completes.

XYZ Widget, Inc. has a shop floor automatic data collection (ADC) system. Each job ticket has a bar code for the assigned job number. Each machine has a sheet with bar codes representing the various operations of that machine. Each employee has a badge with a bar code representing that employee. When an employee starts an operation, the job bar code is scanned, the operation bar code is scanned, and the employee bar code is scanned. The computer pulls in the current system time as the start time. When one operation starts, the previous operation for that employee is automatically stopped (an employee is unable do more than one operation at once). When the work on the widget job is complete on that machine, the employee marks the job complete via the ADC system. The information gathered through the ADC system is used to update scheduling, track the employee's work hours and productivity, and also track the machine's productivity.

The design of a star schema for the scheduling process begins by determining the granularity. The most detailed scheduling table in the operational system has a record for each cost center applicable to manufacturing each job. The users in the scheduling department are interested in drilling down to this level of detail in the data warehouse. The proper level of granularity in the star schema for scheduling is determined by the job number and the cost center.

Next we determine the dimensions in the star schema for the scheduling process. The operational scheduling system tracks the scheduled start and finish date and times, as well as the actual start and finish date and times. The estimated and actual hours are also stored in the operational scheduling details table, along with a flag indicating whether the operation completed on time. The scheduling team must have the ability to group records by the scheduled and actual start and finish times. Also critical is the ability to group by cost center. The dimensions of the star schema for scheduling are the scheduled and actual start and finish date and times, and the cost center. The job number must also be included as a degenerate dimension to maintain the proper granularity in the fact table. Figure 8.9 reflects the decisions on the dimensions appropriate for the scheduling process.

The scheduling team is interested in aggregating the estimated hours and, also, the actual hours. They are also very interested in examining trends in on-time performance. The appropriate measures for the scheduling star schema include the estimated and actual hours and a flag indicating whether the operation was finished on time. The appropriate measures for scheduling are reflected in Figure 8.9.

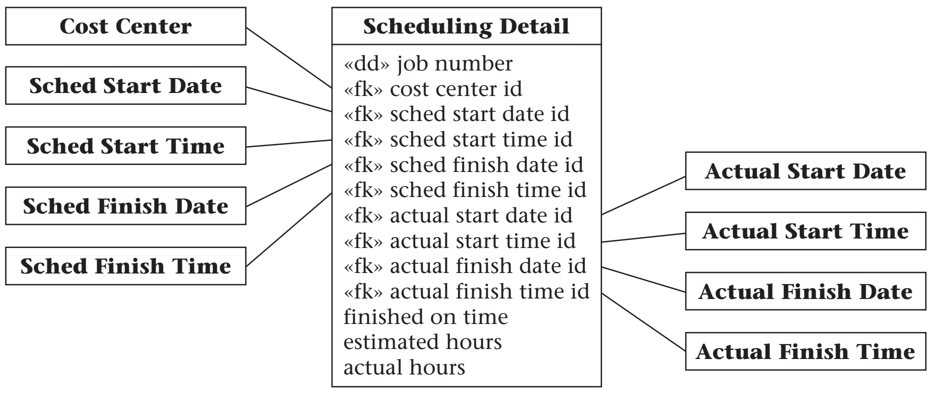


Figure 8.9 Star schema for the scheduling process

There are several standardization principles in play in Figure 8.9. Note that there are multiple time dimensions. These should be standardized with a single time dimension, along with views filling the different roles, similar to the approach used for the date dimensions. Also, notice the Cost Center dimension is present both in the estimating and the scheduling processes. These are actually the same, and should be designed as a single dimension. Dimensions can be shared between multiple star schemas. One last point: the estimated hours are carried from estimating into scheduling in the operational systems. These numbers feed into the star schemas for both the estimating and the scheduling processes. The meaning is the same between the two attributes; therefore, they are both named "estimated hours." The rule of thumb is that if two attributes carry the same meaning, they should be named the same, and if two attributes are named the same, they carry the same meaning. This consistency allows discussion and comparison of information between business processes across the company.

The next process we examine is productivity tracking. The granularity is determined by the level of detail available in the ADC system. The detail includes the job number, cost center, employee number, and the start and finish date and time. The department managers need to be able to group rows by cost center, employee, and start and finish date and times. These attributes therefore become the dimensions of the star schema for the productivity process, shown in Figure 8.10. The managers are interested in aggregating productivity numbers, including the widget quantity produced, the percentage finished on time and the estimated and actual hours. Since these attributes are to be aggregated, they become the measures shown in Figure 8.10.

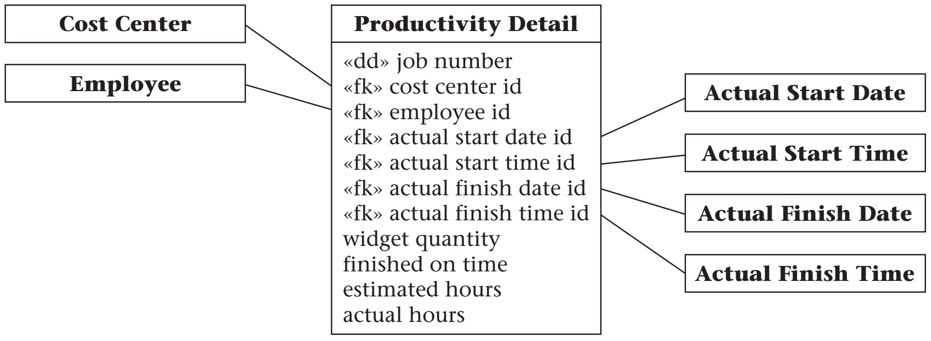


Figure 8.10 Star schema for the productivity tracking process

There are often dimensions in common between star schemas in a data warehouse, because business processes are usually interlinked. A useful tool for tracking the commonality and differences of dimensions across multiple business processes is the data warehouse bus [Kimball and Ross, 2002]. Table 8.2 shows a data warehouse bus for the four business processes in our dimensional design example. Each row represents a business process. Each column represents a dimension. Each X in the body of the table represents the use of the given dimension in the given business process. The data warehouse bus is a handy means of presenting the organization of a data warehouse at a high level. The dimensions common between multiple business processes need to be standardized or "conformed" in Kimball and Ross's terminology. A dimension is conformed if there exists a most detailed version of that dimension, and all other uses of that dimension utilize a subset of the attributes and a subset of the rows from that most detailed version. Conforming dimensions ensures that whenever data are related or compared across business processes, the result is meaningful.

The data warehouse bus also makes some design decisions more obvious. We have taken the liberty of choosing the dimensions for the job-costing process. Table 8.2 includes a row for the job-costing process. When you compare the rows for estimating and job costing, it quickly becomes clear that the two processes have most of the same dimensions. It probably makes sense to combine these two processes into one star

Table 8.2 Data Warehouse Bus for Widget Example

	Shape	*Color*	*Texture*	*Density*	*Size*	*Estimate Date*	*Win Date*	*Customer*	*Promotion*	*Cost Center*	*Sched Start Date*	*Sched Start Time*	*Sched Finish Date*	*Sched Finish Time*	*Actual Start Date*	*Actual Start Time*	*Actual Finish Date*	*Actual Finish Time*	*Employee*	*Invoice Date*
Estimating	X	X	X	X	X	X	X	X	X	X										
Scheduling										X	X	X	X	X	X	X	X	X		
Productivity Tracking										X					X	X	X	X	X	
Job Costing	X	X	X	X	X			X	X	X										X

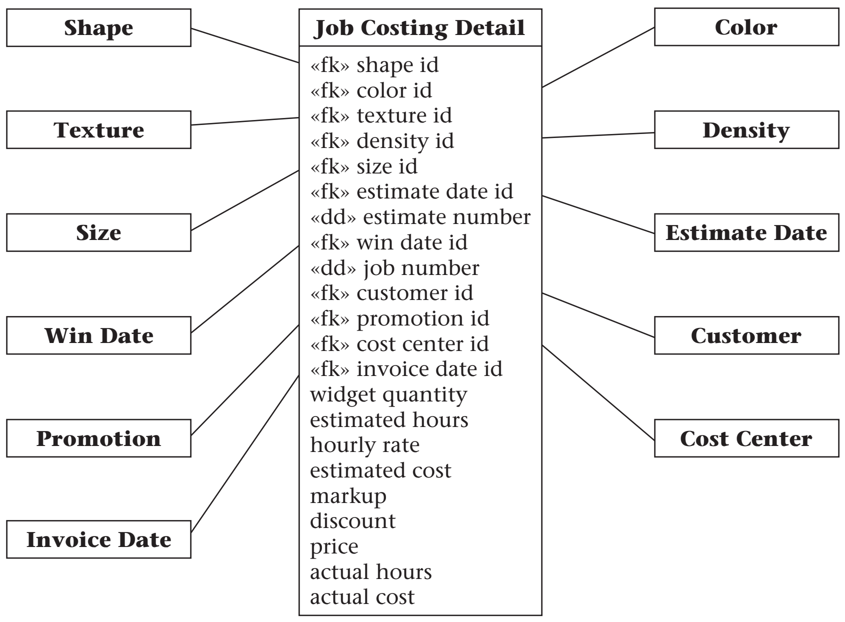


Figure 8.11 Star schema for the job costing process

schema. This is especially true since job-costing analysis requires comparing estimated and actual values. Figure 8.11 is the result of combining the estimating and job costing processes into one star schema.

Summarizing Data

The star schemas we have covered so far are excellent for capturing the pertinent details. Having fine granularity available in the fact table allows the users to examine data down to that level of granularity. However, the users will often want summaries. For example, the managers may often query for a daily snapshot of the job-costing data. Every query the user may wish to pose against a given star schema can be answered from the detailed fact table. The summary could be aggregated on the fly from the fact table. There is an obvious drawback to this strategy. The fact table contains many millions of rows, due to the detailed nature of the data. Producing a summary on the fly can be expensive in terms of computer resources, resulting in a very slow response. If a summary table were available to answer the queries for the job costing daily snapshot, then the answer could be presented to the user blazingly fast.

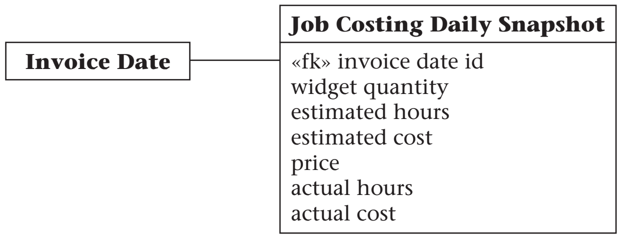


Figure 8.12 Schema for the job costing daily snapshot

The schema for the job costing daily snapshot is shown in Figure 8.12. Notice that most of the dimensions used in the job-costing detail are not used in the snapshot. Summarizing the data has eliminated the need for most dimensions in this context. The daily snapshot contains one row for each day that jobs have been invoiced. The number of rows in the snapshot would be in the thousands. The small size of the snapshot allows very quick response when a user requests the job costing daily snapshot. When there are a small number of summary queries that occur frequently, it is a good strategy to materialize the summary data needed to answer the queries quickly.

The daily snapshot schema in Figure 8.12 also allows the user to group by month, quarter, or year. Materializing summary data is useful for quick response to any query that can be answered by aggregating the data further.

8.2 Online Analytical Processing (OLAP)

Designing and implementing strategic summary tables is a good approach when there is a small set of frequent queries for summary data. However, there may be a need for some users to explore the data in an *ad hoc* fashion. For example, a user who is looking for types of jobs that have not been profitable needs to be able to roll up and drill down various dimensions of the data. The *ad hoc* nature of the process makes predicting the queries impossible. Designing a strategic set of summary tables to answer these *ad hoc* explorations of the data is a daunting task. OLAP provides an alternative. OLAP is a service that overlays the data warehouse. The OLAP system automatically selects a strategic set of summary views, and saves the automatic summary tables (AST) to disk as materialized views. The OLAP system also maintains these views, keeping them in step with the fact tables as new data arrives. When a user requests summary data, the OLAP system figures out which AST can be used for a quick response to the given query. OLAP systems are a good solution when there is a need for *ad hoc* exploration of summary information based on large amounts of data residing in a data warehouse.

OLAP systems automatically select, maintain, and use the ASTs. Thus, an OLAP system effectively does some of the design work automatically. This section covers some of the issues that arise in building an OLAP engine, and some of the possible solutions. If you use an OLAP system, the vendor delivers the OLAP engine to you. The issues and solutions discussed here are not items that you need to resolve. Our goal here is to remove some of the mystery about what an OLAP system is and how it works.

8.2.1 The Exponential Explosion of Views

Materialized views aggregated from a fact table can be uniquely identified by the aggregation level for each dimension. Given a hierarchy along a dimension, let 0 represent no aggregation, 1 represent the first level of aggregation, and so on. For example, if the Invoice Date dimension has a hierarchy consisting of date id, month, quarter, year and "all" (i.e., complete aggregation), then date id is level 0, month is level 1, quarter is level 2, year is level 3, and "all" is level 4. If a dimension does not explicitly have a hierarchy, then level 0 is no aggregation, and level 1 is "all." The scales so defined along each dimension define a coordinate system for uniquely identifying each view in a product graph. Figure 8.13 illustrates a product graph in two dimensions. Product graphs are a generalization of the hypercube lattice structure introduced by Harinarayan, Rajaraman, and Ullman [1996], where dimensions may have associated hierarchies. The top node, labeled (0, 0) in Figure 8.13, represents the fact table. Each node represents a view with aggregation levels as indicated by the coordinate. The relationships descending the product graph indicate aggregation relationships. The five shaded nodes indicate that these views have been materialized. A view can be aggregated from any materialized ancestor view. For example, if a user issues a query for rows grouped by year and state, that query would naturally be answered by the view labeled (3, 2). View (3, 2) is not materialized, but the query can be answered from the materialized view (2, 1) since (2, 1) is an ancestor of (3, 2). Quarters can be aggregated into years, and cities can be aggregated into states.

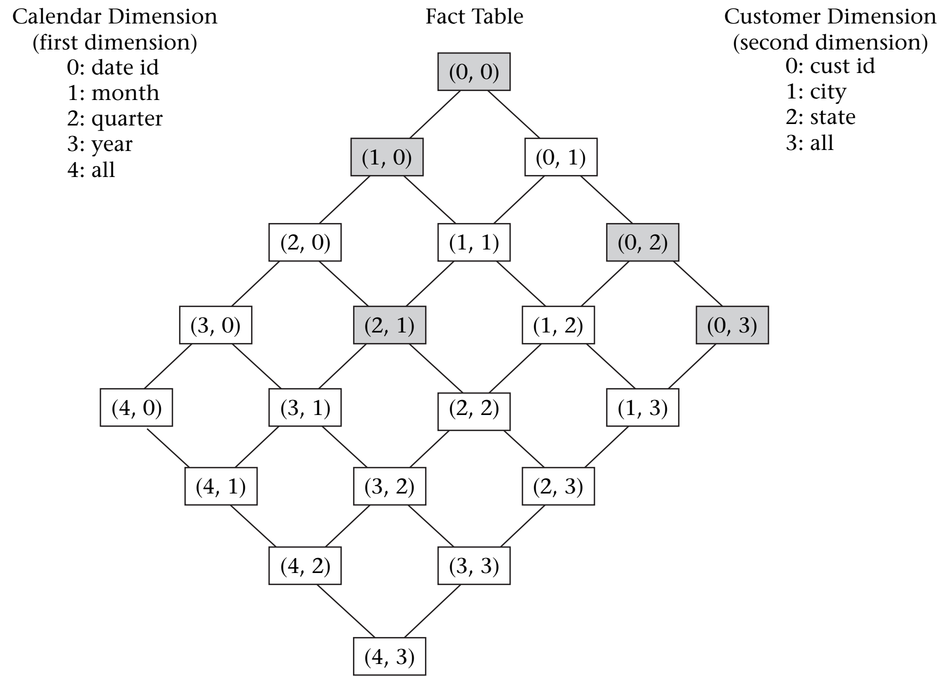


Figure 8.13 Product graph labeled with aggregation level coordinates

The central issue challenging the design of OLAP systems is the exponential explosion of possible views as the number of dimensions increases. The Calendar dimension in Figure 8.13 has five levels of hierarchy, and the Customer dimension has four levels of hierarchy. The user may choose any level of aggregation along each dimension. The number of possible views is the product of the number of hierarchical levels along each dimension. The number of possible views for the example in Figure 8.13 is $5 \times 4 = 20$. Let d be the number of dimensions in a data warehouse. Let h_i be the number of hierarchical levels in dimension i. The general equation for calculating the number of possible views is given by Equation 8.1.

$$\text{Possible views} = \prod_{i=1}^{d} h_i \qquad 8.1$$

If we express Equation 8.1 in different terms, the problem of exponential explosion becomes more apparent. Let g be the geometric mean of the number of hierarchical levels in the dimensions. Then Equation 8.1 becomes Equation 8.2.

$$\text{Possible views} = g^d \qquad 8.2$$

As dimensionality increases linearly, the number of possible views explodes exponentially. If $g = 5$ and $d = 5$, there are $5^5 = 3{,}125$ possible views. Thus if $d = 10$, then there are $5^{10} = 9{,}765{,}625$ possible views. OLAP administrators need the freedom to scale up the dimensionality of their data warehouses. Clearly the OLAP system cannot create and maintain all possible views as dimensionality increases. The design of OLAP systems must deliver quick response while maintaining a system within the resource limitations. Typically, a strategic subset of views must be selected for materialization.

8.2.2 Overview of OLAP

There are many approaches to implementing OLAP systems presented in the literature. Figure 8.14 maps out one possible approach, which will serve for discussion. The larger problem of OLAP optimization is broken into four subproblems: view size estimation, materialized view selection, materialized view maintenance, and query optimization with materialized views. This division is generally true of the OLAP literature, and is reflected in the OLAP system plan shown in Figure 8.14.

We describe how the OLAP processes interact in Figure 8.14, and then explore each process in greater detail. The plan for OLAP optimization shows *Sample Data* moving from the *Fact Table* into *View Size Estimation*. *View Selection* makes an *Estimate Request* for the view size of each view it considers for materialization. *View Size Estimation* queries the *Sample Data*, examines it, and models the distribution. The distribution observed in the sample is used to estimate the expected number of rows in the view for the full dataset. The *Estimated View Size* is passed to *View Selection*, which uses the estimates to evaluate the relative benefits of materializing the various views under consideration. *View Selection* picks *Strategically Selected Views* for materialization with the goal of minimizing total query costs. *View Maintenance* builds the original views from the *Initial Data* from the *Fact Table*, and maintains the views as *Incremental Data* arrives from *Updates*. *View Maintenance* sends statistics on *View Costs* back to *View Selection*, allowing costly views to be discarded dynamically. *View Maintenance* offers *Current Views* for use by *Query Optimization*. *Query Optimization* must consider which of the *Current Views*

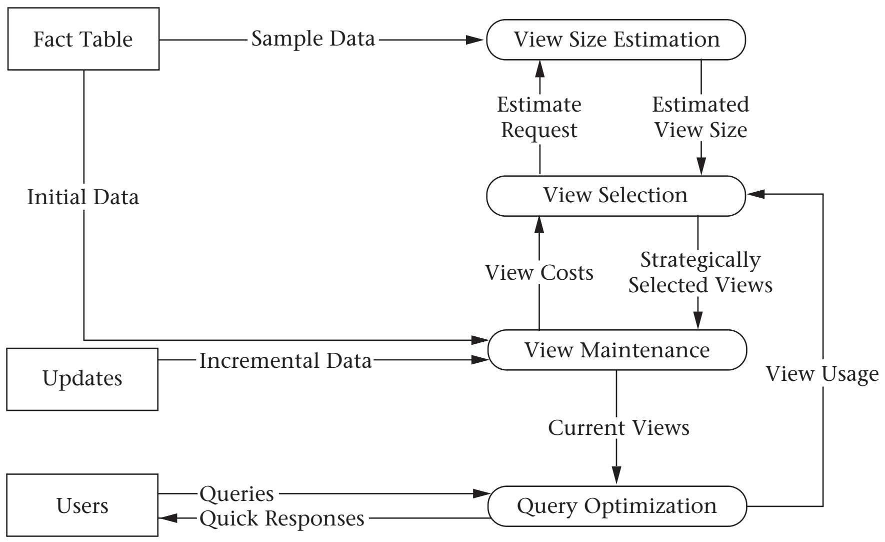


Figure 8.14 A plan for OLAP optimization

can be utilized to most efficiently answer *Queries* from *Users*, giving *Quick Responses* to the *Users*. *View Usage* feeds back into *View Selection*, allowing the system to dynamically adapt to changes in query workloads.

8.2.3 View Size Estimation

OLAP systems selectively materialize strategic views with high benefits to achieve quick response to queries, while remaining within the resource limits of the computer system. The size of a view affects how much disk space is required to store the view. More importantly, the size of the view determines in part how much disk input/output will be consumed when querying and maintaining the view. Calculating the exact size of a given view requires calculating the view from the base data. Reading the base data and calculating the view is the majority of the work necessary to materialize the view. Since the objective of view materialization is to conserve resources, it becomes necessary to estimate the size of the views under consideration for materialization.

Cardenas' formula [Cardenas, 1975] is a simple equation (Equation 8.3) that is applicable to estimating the number of rows in a view:

Let n be the number of rows in the fact table.

Let v be the number of possible keys in the data space of the view.

$$\text{Expected distinct values} = v(1 - (1 - 1/v)^n) \qquad 8.3$$

Cardenas' formula assumes a uniform data distribution. However, many data distributions exist. The data distribution in the fact table affects the number of rows in a view. Cardenas' formula is very quick, but the assumption of a uniform data distribution leads to gross overestimates of the view size when the data is actually clustered. Other methods have been developed to model the effect of data distribution on the number of rows in a view.

Faloutsos, Matias, and Silberschatz [1996] present a sampling approach based on the binomial multifractal distribution. Parameters of the distribution are estimated from a sample. The number of rows in the aggregated view for the full data set is then estimated using the parameter values determined from the sample. Equations 8.4 and 8.5 [Faloutsos, Matias, and Silberschatz, 1996] are presented for this purpose.

$$\text{Expected distinct values} = \sum_{a=0}^{k} C_a^k (1 - (1 - P_a)^n) \qquad 8.4$$

$$P_a = P^{k-a}(1 - P)^a \qquad 8.5$$

Figure 8.15 illustrates an example. Order k is the decision tree depth. C_a^k is the number of bins in the set reachable by taking some combination of a left hand edges and $k - a$ right hand edges in the decision tree. P_a is the probability of reaching a given bin whose path contains a left hand edges. n is the number of rows in the data set. Bias P is the probability of selecting the right hand edge at a choice point in the tree.

The calculations of Equation 8.4 are illustrated with a small example. An actual database would yield much larger numbers, but the concepts and the equations are the same. These calculations can be done with logarithms, resulting in very good scalability. Based on Figure 8.15, given five rows, calculate the expected distinct values using Equation 8.4:

$$\begin{aligned}&\text{Expected distinct values} =\\ &1 \cdot (1 - (1 - 0.729)^5) + 3 \cdot (1 - (1 - 0.081)^5) +\\ &3 \cdot (1 - (1 - 0.009)^5) + 1 \cdot (1 - (1 - 0.001)^5) \approx 1.965 \qquad 8.6\end{aligned}$$

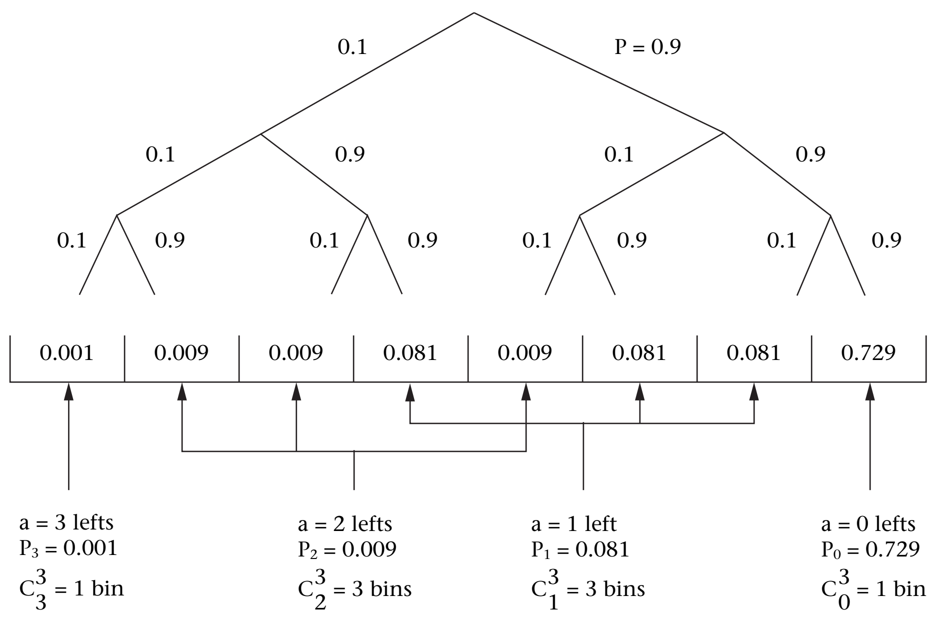


Figure 8.15 Example of a binomial multifractal distribution tree

The values of *P* and *k* can be estimated based on sample data. The algorithm used in [Faloutsos, Matias, and Silberschatz, 1996] has three inputs: the number of rows in the sample, the frequency of the most commonly occurring value, and the number of distinct aggregate rows in the sample. The value of *P* is calculated based on the frequency of the most commonly occurring value. They begin with:

$$k = \lceil \text{Log}_2(\text{Distinct rows in sample}) \rceil \qquad 8.7$$

and then adjust *k* upwards, recalculating *P* until a good fit to the number of distinct rows in the sample is found.

Other distribution models can be utilized to predict the size of a view based on sample data. For example, the use of the Pareto distribution model has been explored [Nadeau and Teorey, 2003]. Another possibility is to find the best fit to the sample data for multiple distribution models, calculate which model is most likely to produce the given sample data, and then use that model to predict the number of rows for the full data set. This would require calculation for each distribution model considered, but should generally result in more accurate estimates.

8.2.4 Selection of Materialized Views

Most of the published works on the problem of materialized view selection are based on the hypercube lattice structure [Harinarayan, Rajaraman, and Ullman, 1996]. The hypercube lattice structure is a special case of the product graph structure, where the number of hierarchical levels for each dimension is two. Each dimension can either be included or excluded from a given view. Thus, the nodes in a hypercube lattice structure represent the power set of the dimensions.

Figure 8.16 illustrates the hypercube lattice structure with an example [Harinarayan, Rajaraman, and Ullman, 1996]. Each node of the lattice structure represents a possible view. Each node is labeled with the set of dimensions in the "group by" list for that view. The numbers associated with the nodes represent the number of rows in the view. These numbers are normally derived from a view size estimation algorithm, as discussed in Section 8.2.3. However, the numbers in Figure 8.16 follow the example as given by Harinarayan et al. [1996]. The relationships between nodes indicate which views can be aggregated from other views. A given view can be calculated from any materialized ancestor view.

We refer to the algorithm for selecting materialized views introduced by Harinarayan et al. [1996] as HRU. The initial state for HRU has only the fact table materialized. HRU calculates the benefit of each possible view during each iteration, and selects the most beneficial view for materialization. Processing continues until a predetermined number of materialized views is reached.

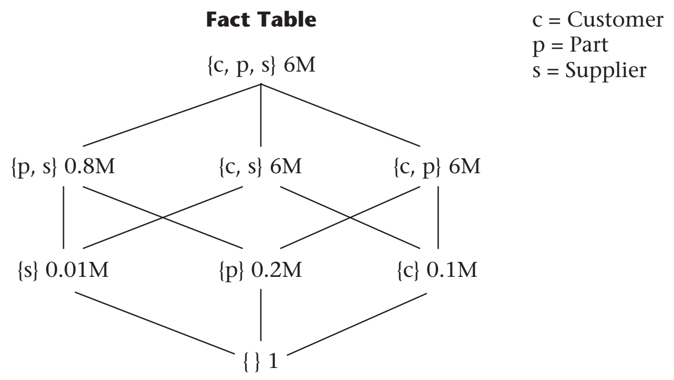


Figure 8.16 Example of a hypercube lattice structure [Harinarayan et al. 1996]

Table 8.3 Two Iterations of HRU, Based on Figure 8.16

	Iteration 1 Benefit	*Iteration 2 Benefit*
{p, s}	**5.2M × 4 = 20.8M**	
{c, s}	0 × 4 = 0	0 × 2 = 0
{c, p}	0 × 4 = 0	0 × 2 = 0
{s}	5.99M × 2 = 11.98M	0.79M × 2 = 1.58M
{p}	5.8M × 2 = 11.6M	0.6M × 2 = 1.2M
{c}	5.9M × 2 = 11.8M	**5.9M × 2 = 11.8M**
{}	6M – 1	0.8M – 1

Table 8.3 shows the calculations for the first two iterations of HRU. Materializing {*p*, *s*} saves 6M – 0.8M = 5.2M rows for each of four views: {*p*, *s*} and its three descendants: {p}, {s}, and {}. The view {*c*, *s*} yields no benefit materialized, since any query that can be answered by reading 6M rows from {*c*, *s*} can also be answered by reading 6M rows from the fact table {*c*, *p*, *s*}. HRU calculates the benefits of each possible view materialization. The view {*p*, *s*} is selected for materialization in the first iteration. The view {*c*} is selected in the second iteration.

HRU is a greedy algorithm that does not guarantee an optimal solution, although testing has shown that it usually produces a good solution. Further research has built upon HRU, accounting for the presence of index structures, update costs, and query frequencies.

HRU evaluates every unselected node during each iteration, and each evaluation considers the effect on every descendant. The algorithm consumes $O(kn^2)$ time, where k = |views to select| and n = |nodes|. This order of complexity looks very good; it is polynomial time. However, the result is misleading. The nodes of the hypercube lattice structure constitute a power set. The number of possible views is therefore 2^d where d = |dimensions|. Thus, $n = 2^d$, and the time complexity of HRU is $O(k2^{2d})$. HRU runs in time exponentially relative to the number of dimensions in the database.

The Polynomial Greedy Algorithm (PGA) [Nadeau and Teorey, 2002] offers a more scalable alternative to HRU. PGA, like HRU, also selects one view for materialization with each iteration. However, PGA divides each iteration into a nomination phase and a selection phase. The first phase nominates promising views into a candidate set. The second phase estimates the benefits of materializing each candidate, and selects the view with the highest evaluation for materialization.

Table 8.4 First Iteration of PGA, Based on Figure 8.16

Candidates	*Iteration 1 Benefit*
{p, s}	**5.2M × 4 = 20.8M**
{s}	5.99M × 2 = 11.98M
{}	6M – 1

The nomination phase begins at the top of the lattice; in Figure 8.16, this is the node {*c*, *p*, *s*}. PGA nominates the smallest node from amongst the children. The candidate set is now {{*p*, *s*}}. PGA then examines the children of {*p*, *s*} and nominates the smallest child, {*s*}. The process repeats until the bottom of the lattice is reached. The candidate set is then {{*p*, *s*}, {*s*}, {}}. Once a path of candidate views has been nominated, the algorithm enters the selection phase. The resulting calculations are shown in Tables 8.4 and 8.5.

Compare Tables 8.4 and 8.5 with Table 8.3. Notice PGA does fewer calculations than HRU, and yet in this example reaches the same decisions as HRU. PGA usually picks a set of views nearly as beneficial as those chosen by HRU, and yet PGA is able to function when HRU fails due to the exponential complexity. PGA is polynomial relative to the number of dimensions. When HRU fails, PGA extends the usefulness of the OLAP system.

The materialized view selection algorithms discussed so far are static; that is, the views are picked once and then materialized. An entirely different approach to the selection of materialized views is to treat the problem similar to memory management [Kotidis and Roussopoulos, 1999]. The materialized views constitute a view pool. Metadata is tracked on usage of the views. The system monitors both space and update window constraints. The contents of the view pool are adjusted dynamically. As queries are posed, views are added appropriately. Whenever a constraint is violated, the system selects a view for eviction. Thus the

Table 8.5 Second Iteration of PGA, Based on Figure 8.16

Candidates	*Iteration 2 Benefit*
{c, s}	0 × 2 = 0
{s}	0.79M × 2 = 1.58M
{c}	**5.9M × 2 = 11.8M**
{}	6M – 1

view pool can improve as more usage statistics are gathered. This is a self-tuning system that adjusts to changing query patterns.

The static and dynamic approaches complement each other and should be integrated. Static approaches run fast from the beginning, but do not adapt. Dynamic view selection begins with an empty view pool, and therefore yields slow response times when a data warehouse is first loaded; however, it is adaptable and improves over time. The complementary nature of these two approaches has influenced our design plan in Figure 8.14, as indicated by *Queries* feeding back into *View Selection*.

8.2.5 View Maintenance

Once a view is selected for materialization, it must be computed and stored. When the base data is updated, the aggregated view must also be updated to maintain consistency between views. The original view materialization and the incremental updates are both considered as view maintenance in Figure 8.14. The efficiency of view maintenance is greatly affected by the data structures implementing the view. OLAP systems are multidimensional, and fact tables contain large numbers of rows. The access methods implementing the OLAP system must meet the challenges of high dimensionality in combination with large row counts. The physical structures used are deferred to volume two, which covers physical design.

Most of the research papers in the area of view maintenance assume that new data is periodically loaded with incremental data during designated update windows. Typically, the OLAP system is made unavailable to the users while the incremental data is loaded in bulk, taking advantage of the efficiencies of bulk operations. There is a down side to deferring the loading of incremental data until the next update window. If the data warehouse receives incremental data once a day, then there is a one-day latency period.

There is currently a push in the industry to accommodate data updates close to real time, keeping the data warehouse in step with the operational systems. This is sometimes referred to as “active warehousing” and “real-time analytics.” The need for data latency of only a few minutes presents new problems. How can very large data structures be maintained efficiently with a trickle feed? One solution is to have a second set of data structures with the same schema as the data warehouse. This second set of data structures acts as a holding tank for incremental data, and is referred to as a delta cube in OLAP terminology. The operational systems feed into the delta cube, which is small and efficient for quick incremental changes. The data cube is updated periodically from the delta cube, taking advantage of bulk operation efficiencies. When the user queries the OLAP system, the query can be issued against both the data cube and the delta cube to obtain an up-to-date result. The delta cube is hidden from the user. What the user sees is an OLAP system that is nearly current with the operational systems.

8.2.6 Query Optimization

When a query is posed to an OLAP system, there may be multiple materialized views available that could be used to compute the result. For example, if we have the situation represented in Figure 8.13, and a user issues a query to group rows by month and state, that query is naturally answered from the view labeled (1, 2). However, since (1, 2) is not materialized, we need to find a materialized ancestor to obtain the data. There are three such nodes in the product graph of Figure 8.13. The query can be answered from nodes (0, 0), (1, 0), or (0, 2). With the possibility of answering queries from alternative sources, the optimization issue arises as to which source is the most efficient for the given query. Most existing research focuses on syntactic approaches. The possible query translations are carried out, alternative query costs are estimated, and what appears to be the best plan is executed. Another approach is to query a metadata table containing information on the materialized views to determine the best view to query against, and then translate the original SQL query to use the best view.

Database systems contain metadata tables that hold data about the tables and other structures used by the system. The metadata tables facilitate the system in its operations. Here's an example where a metadata

Table 8.6 Example of Materialized View Metadata

Dimensions			
Calendar	*Customer*	*Blocks*	*ViewID*
0	0	10,000,000	1
0	2	50,000	3
0	3	1,000	5
1	0	300,000	2
2	1	10,000	4

table can facilitate the process of finding the best view to answer a query in an OLAP system. The coordinate system defined by the aggregation levels forms the basis for organizing the metadata for tracking the materialized views. Table 8.6 displays the metadata for the materialized views shaded in Figure 8.13. The two dimensions labeled *Calendar* and *Customer* form the composite key. The *Blocks* column tracks the actual number of blocks in each materialized view. The *ViewID* column is used to identify the associated materialized view. The implementation stores materialized views as tables where the value of the *ViewID* forms part of the table name. For example, the row with *ViewID* = 3 contains information on the aggregated view that is materialized as table **AST3** (short for automatic summary table 3).

Observe the general pattern in the coordinates of the views in the product graph with regard to ancestor relationships. Let Value(V, d) represent a function that returns the aggregation level for view V along dimension d. For any two views V_i and V_j where $V_i \neq V_j$, V_i is an ancestor of V_j if and only if for every dimension d of the composite key, Value(V_i, d) $\leq$ Value(V_j, d). This pattern in the keys can be utilized to identify ancestors of a given view by querying the metadata. The semantics of the product graph are captured by the metadata, permitting the OLAP system to search semantically for the best materialized ancestor view by querying the metadata table. After the best materialized view is determined, the OLAP system can rewrite the original query to utilize the best materialized view, and proceed.

8.3 Data Mining

Two general approaches are used to extract knowledge from a database. First, a user may have a hypothesis to verify or disprove. This type of analysis is done with standard database queries and statistical analysis. The second approach to extracting knowledge is to have the computer search for correlations in the data, and present promising hypotheses to the user for consideration. The methods included here are data mining techniques developed in the fields of Machine Learning and Knowledge Discovery.

Data mining algorithms attempt to solve a number of common problems. One general problem is categorization: given a set of cases with known values for some parameters, classify the cases. For example, given observations of patients, suggest a diagnosis. Another general problem type is clustering: given a set of cases, find natural groupings of the cases. Clustering is useful, for example, in identifying market segments. Association rules, also known as market basket analyses, are another common problem. Businesses sometimes want to know what items are frequently purchased together. This knowledge is useful, for example, when decisions are made about how to lay out a grocery store. There are many types of data mining available. Han and Kamber [2001] cover data mining in the context of data warehouses and OLAP systems. Mitchell [1997] is a rich resource, written from the machine learning perspective. Witten and Frank [2000] give a survey of data mining, along with freeware written in Java available from the Weka Web site [http://www.cs.waikato.ac.nz/ml/weka]. The Weka Web site is a good option for those who wish to experiment with and modify existing algorithms. The major database vendors also offer data mining packages that function with their databases.

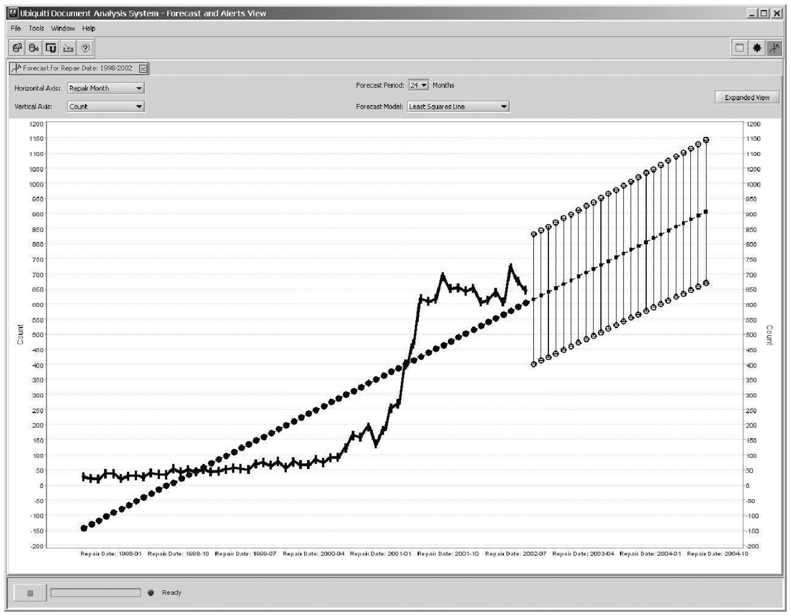
Due to the large scope of data mining, we focus on two forms of data mining: forecasting and text mining.

8.3.1 Forecasting

Forecasting is a form of data mining in which trends are modeled over time using known data, and future trends are predicted based on the model. There are many different prediction models with varying levels of sophistication. Perhaps the simplest model is the least squares line model. The best fit line is calculated from known data points using the method of least squares. The line is projected into the future to determine predictions. Figure 8.17 shows a least squares line for an actual data set. The crossed (jagged) points represent actual known data. The circular (dots) points represent the least squares line. When the least squares line projects beyond the known points, this region represents predictions. The intervals associated with the predictions in our figures represent a 90% prediction interval. That is, given an interval, there is a 90% probability that the actual value, when known, will lie in that interval.

The least squares line approach weights each known data point equally when building the model. The predicted upward trend in Figure 8.17 does not give any special consideration to the recent downturn.

Exponential smoothing is an approach that weights recent history more heavily than distant history. Double exponential smoothing models two components: level and trend (hence "double" exponential smoothing). As the known values change in level and trend, the model adapts. Figure 8.18 shows the predictions made using double exponen-


Ubiquiti Document Analysis System - Forecast and Alerts View
File Tools Window Help
Forecast for Repair Date: 1998-2002
Expanded View
Count
Ready

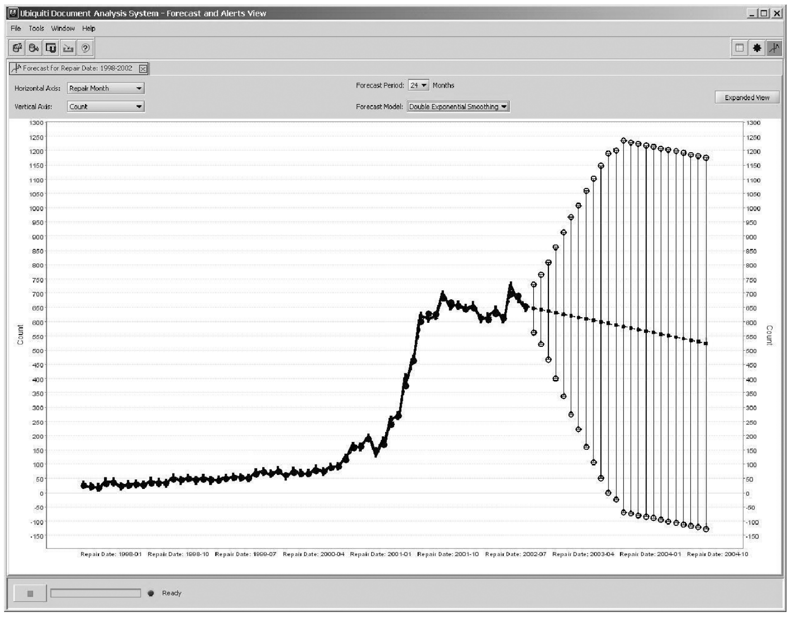
Horizontal Axis: Repair Month
Vertical Axis: Count
Forecast Period: 24 Months
Forecast Model: Least Squares Line

Figure 8.17 Least squares line (courtesy of Ubiquiti, Inc.)

tial smoothing, based on the same data set used to compute Figure 8.17. Notice the prediction is now more tightly bound to recent history.

Triple exponential smoothing models three components: level, trend, and seasonality. This is more sophisticated than double exponential smoothing, and gives better predictions when the data does indeed exhibit seasonal behavior. Figure 8.19 shows the predictions made by triple exponential smoothing, based on the same data used to compute Figures 8.17 and 8.18. Notice the prediction intervals are tighter than in Figures 8.17 and 8.18. This is a sign that the data varies seasonally; triple exponential smoothing is a good model for the given type of data.

Exactly how reliable are these predictions? If we revisit the predictions after time has passed and compare the predictions with the actual values, are they accurate? Figure 8.20 shows the actual data overlaid with the predictions made in Figure 8.19. Most of the actual data points do indeed lie within the prediction intervals. The prediction intervals look very reasonable. Why don't we use these forecast models to make our millions on Wall Street? Take a look at Figure 8.21, a cautionary tale. Figure 8.21 is also based on the triple exponential smoothing model, using four years of known data for training, compared with five years of data used in constructing the model for Figure 8.20. The resulting pre-


Ubiquiti Document Analysis System - Forecast and Alerts View
File
Tools
Window
Help
Forecast for Repair Date: 1998-2002
Expanded View
Count
Ready

Horizontal Axis: Repair Month
Vertical Axis: Count
Forecast Period: 24 Months
Forecast Model: Double Exponential Smoothing

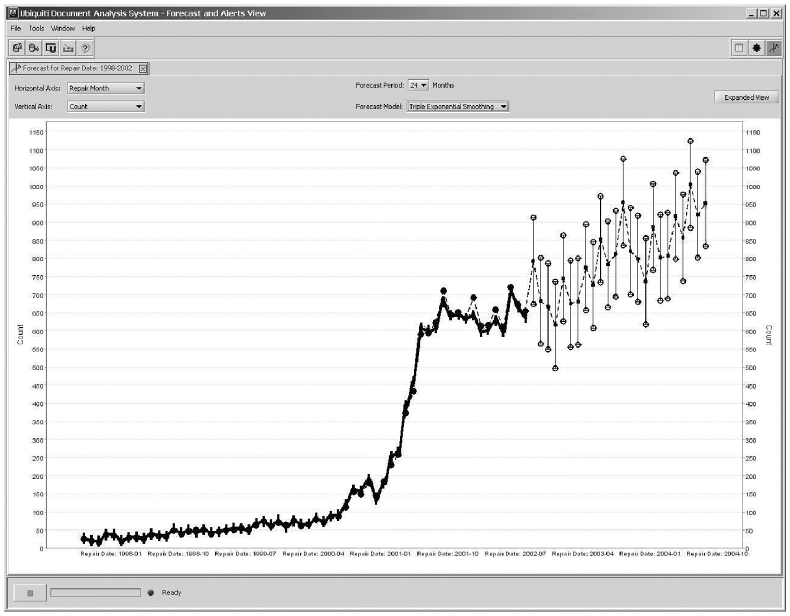
Figure 8.18 Double exponential smoothing (courtesy of Ubiquiti, Inc.)

dictions match for four months, and then diverge greatly from reality. The problem is that forecast models are built on known data, with the assumption that known data forms a good basis for predicting the future. This may be true most of the time; however, forecast models can be unreliable when the market is changing or about to change drastically. Forecasting can be a useful tool, but the predictions must be taken only as indicators.

The details of the forecast models discussed here, as well as many others, can be found in Makridakis et al. [1998].

8.3.2 Text Mining

Most of the work on data processing over the past few decades has used structured data. The vast majority of systems in use today read and store data in relational databases. The schemas are organized neatly in rows and columns. However, there are large amounts of data that reside in freeform text. Descriptions of warranty claims are written in text. Medical records are written in text. Text is everywhere. Only recently has the work in text analysis made significant headway. Companies are now marketing products that focus on text analysis.


Ubiquiti Document Analysis System - Forecast and Alerts View

Forecast for Repair Date: 1998-2002

Horizontal Axis: Repair Month

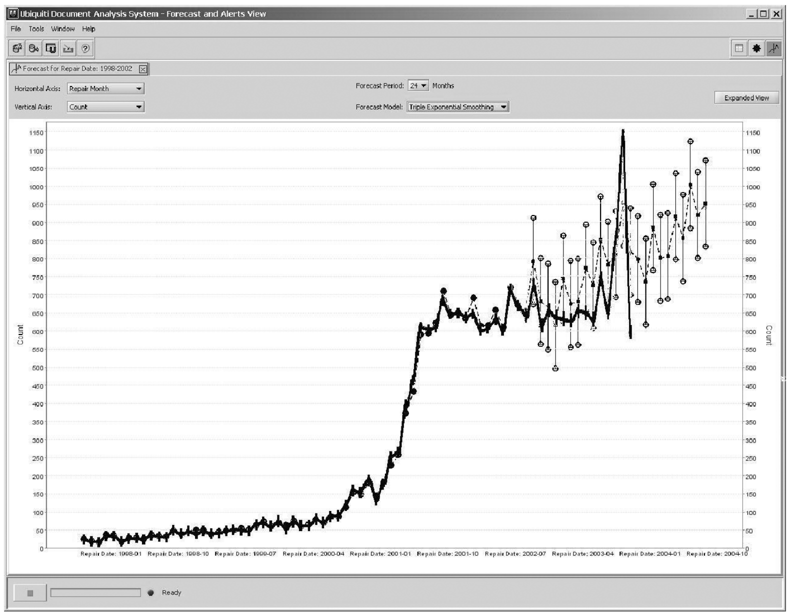
Vertical Axis: Count

Forecast Period: 24 Months

Forecast Model: Triple Exponential Smoothing

Ready

Figure 8.19 Triple exponential smoothing (courtesy of Ubiquiti, Inc.)


Ubiquiti Document Analysis System - Forecast and Alerts View

Forecast for Repair Date: 1998-2002

Horizontal Axis: Repair Month

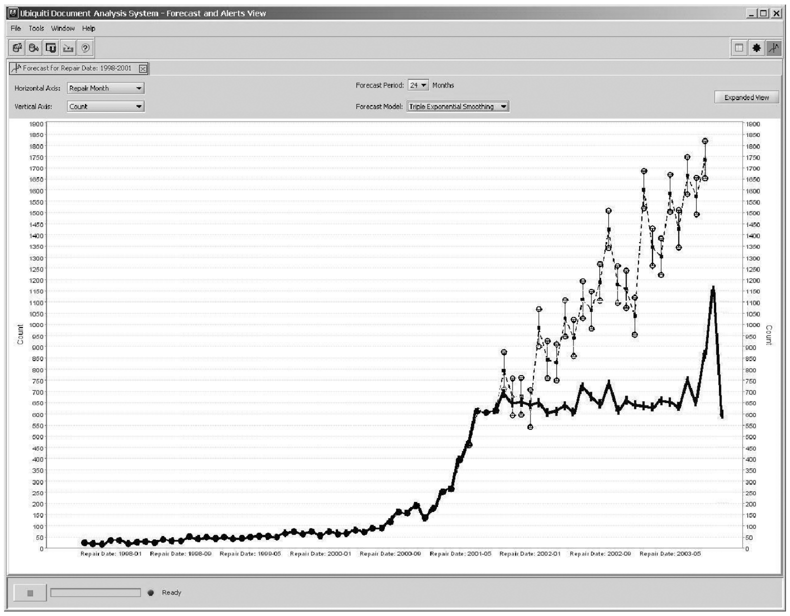
Vertical Axis: Count

Forecast Period: 24 Months

Forecast Model: Triple Exponential Smoothing

Ready

Figure 8.20 Triple exponential smoothing with actual values overlaying forecast values, based on five years of training data (courtesy of Ubiquiti, Inc.)


Ubiquiti Document Analysis System - Forecast and Alerts View

Forecast for Repair Date: 1998-2001

Horizontal Axis: Repair Month

Vertical Axis: Count

Forecast Period: 24 Months

Forecast Model: Triple Exponential Smoothing

Figure 8.21 Triple exponential smoothing with actual values overlaying forecast values, based on four years of training data (courtesy of Ubiquiti, Inc.)

Let's look at a few of the possibilities for analyzing text and their potential impact. We'll take the area of automotive warranty claims as an example. When something goes wrong with your car, you bring it to an automotive shop for repairs. You describe to a shop representative what you've observed going wrong with your car. Your description is typed into a computer. A mechanic works on your car, and then types in observations about your car and the actions taken to remedy the problem. This is valuable information for the automotive companies and the parts manufacturers. If the information can be analyzed, they can catch problems early and build better cars. They can reduce breakdowns, saving themselves money, and saving their customers frustration.

The data typed into the computer is often entered in a hurry. The language includes abbreviations, jargon, misspelled words, and incorrect grammar. Figure 8.22 shows an example entry from an actual warranty claim database.

As you can see, the raw information entered on the shop floor is barely English. Figure 8.23 shows a cleaned up version of the same text.

7 DD40 BASC 54566 CK OUT AC INOP PREFORM PID CK CK PCM PID ACC CK OK OPERATING ON AND OFF PREFORM POWER AND GRONED CK AT COMPRESOR FONED NO GRONED PREFORM PINPONT DIAG AND TRACE GRONED FONED BAD CO NECTION AT S778 REPAIR AND RETEST OK CK AC OPERATION

Figure 8.22 Example of a verbatim description in a warranty claim (courtesy of Ubiquiti, Inc.)

7 DD40 Basic 54566 Check Out Air Conditioning Inoperable Perform PID Check Check Power Control Module PID Accessory Check OK Operating On And Off Perform Power And Ground Check At Compressor Found No Ground Perform Pinpoint Diagnosis And Trace Ground Found Bad Connection At Splice 778 Repair And Retest OK Check Air Conditioning Operation.

Figure 8.23 Cleaned up version of description in warranty claim (courtesy of Ubiquiti, Inc.)

Even the cleaned up version is difficult to read. The companies paying out warranty claims want each claim categorized in various ways, to track what problems are occurring. One option is to hire many people to read the claims and determine how each claim should be categorized. Categorizing the claims manually is tedious work. A more viable option, developed in the last few years, is to apply a software solution. Figure 8.24 shows some of the information that can be gleaned automatically from the text in Figure 8.22.

The software processes the text and determines the concepts likely represented in the text. This is not a simple word search. Synonyms map

Automated Coding	Confidence
Primary Group: Electrical	90 %
Subgroup: Climate Control	85 %
Part: Connector 1008	93 %
Problem: Bad Connection	72 %
Repair: Reconnect	75 %
Location: Engin. Cmprt.	90 %

Figure 8.24 Useful information extracted from verbatim description in warranty claim (courtesy of Ubiquiti, Inc.)

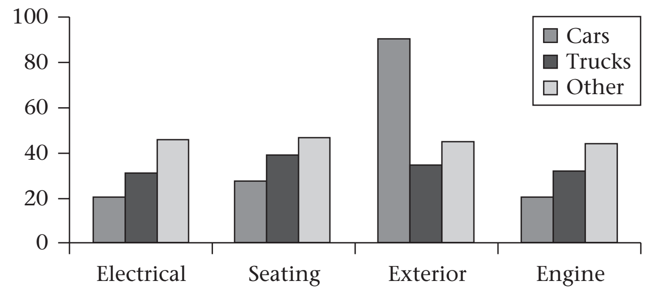


Figure 8.25 Aggregate data from warranty claims (courtesy of Ubiquiti, Inc.)

to the same concept. Some words map to different concepts depending on the context. The software uses an ontology that relates words and concepts to each other. After each warranty is categorized in various ways, it becomes possible to obtain useful aggregate information, as shown in Figure 8.25.

8.4 Summary

Data warehousing, OLAP, and data mining are three areas of computer science that are tightly interlinked and marketed under the heading of business intelligence. The functionalities of these three areas complement each other. Data warehousing provides an infrastructure for storing and accessing large amounts of data in an efficient and user-friendly manner. Dimensional data modeling is the approach best suited for designing data warehouses. OLAP is a service that overlays the data warehouse. The purpose of OLAP is to provide quick response to *ad hoc* queries, typically involving grouping rows and aggregating values. Roll-up and drill-down operations are typical. OLAP systems automatically perform some design tasks, such as selecting which views to materialize in order to provide quick response times. OLAP is a good tool for exploring the data in a human-driven fashion, when the person has a clear question in mind. Data mining is usually computer driven, involving analysis of the data to create likely hypotheses that might be of interest to users. Data mining can bring to the forefront valuable and interesting structure in the data that would otherwise have gone unnoticed.

8.5 Literature Summary

The evolution and principles of data warehouses can be found in Barquin and Edelstein [1997], Cataldo [1997], Chaudhuri and Dayal [1997], Gray and Watson [1998], Kimball and Ross [1998, 2002], and Kimball and Caserta [2004]. OLAP is discussed in Barquin and Edelstein [1997], Faloutsos, Matia, and Silberschatz [1996], Harinarayan, Rajaraman, and Ullman [1996], Kotidis and Roussopoulos [1999], Nadeau and Teorey [2002 2003], Thomsen [1997], and data mining principles and tools can be found in Han and Kamber [2001], Makridakis, Wheelwright, and Hyndman [1998], Mitchell [1997], The University of Waikato [2005], Witten and Frank [2000], among many others.