

CHAPTER 11

DATA STORAGE DESIGN

Another important activity of the design phase is designing the data storage component of the system. This chapter describes the activities that are performed when developing the data storage design. First, the different ways in which data can be stored are described. Several important characteristics that should be considered when selecting the data storage format are discussed. The process of revising the logical data model into the physical data model is then outlined. Because one of the most popular data storage formats today is the relational database, optimization of relational databases from both storage and access perspectives is also included.

OBJECTIVES

- Become familiar with several file and database formats.
- Describe several goals of data storage.
- Be able to revise a logical ERD into a physical ERD.
- Be able to optimize a relational database for data storage and data access.
- Become familiar with indexes.
- Be able to estimate the size of a database.

CHAPTER OUTLINE

Introduction

Data Storage Formats

Files

Databases

Selecting a Storage Format

Applying the Concepts at Tune Source

Moving from Logical to Physical Data

Models

*The Physical Entity Relationship
Diagram*

Revisiting the CRUD Matrix

Applying the Concepts at Tune Source

Optimizing Data Storage

Optimizing Storage Efficiency

Optimizing Access Speed

Estimating Storage Size

Applying the Concepts at Tune Source

Summary

INTRODUCTION

As explained in Chapter 8, the work done by any application program can be divided into four general functions: data storage, data access logic, application logic, and presentation logic. The data storage function is concerned with how data is stored and handled by the programs that run the system.

Applications are of little use without the data that they support. How useful is a multimedia application that can't support images or sound? Why would users log into a system to find information if it took them less time to locate the information manually? It is essential to ensure that data storage is designed so that inefficient systems, long response times, and difficult-to-access information (several hallmarks of failed systems) are avoided.

As analysts turn their attention to the data storage that will be needed for the new system, several things must be done. First, the data storage format for the new system must be selected. This chapter describes a variety of data storage formats and explains how to select the appropriate one for your application. There are two basic types of data storage formats for application systems: files and databases. There are multiple types of each storage format; for example, databases can be object-oriented, relational, multidimensional, and so on. Each type has certain characteristics that make it preferable for certain situations.

Following the selection of the data storage format, the data model created during analysis is modified to reflect this implementation decision. The logical data model will be converted into a *physical data model*. CASE repository information is expanded to include much more detailed information about specific implementation details. The analysts will also want to ensure that the DFDs and ERDs balance properly, so the CRUD matrix will be revised as necessary.

Finally, the selected data storage format must be designed to optimize its processing efficiency. One of the leading complaints by end users is that the final system is too slow. To avoid such complaints, project team members must allow time during the design phase to carefully make sure that the file or database performs as fast as possible. At the same time, the team must keep hardware costs down by minimizing the storage space that the application will require. The goals of maximizing access to data and minimizing the amount of space taken to store data can conflict, so designing data storage efficiency usually requires trade-offs. The team must carefully review the availability, reliability, and security nonfunctional requirements to identify issues that produce trade-offs in performance, cost, and storage space.

DATA STORAGE FORMATS

There are two main types of data storage formats: files and databases. *Files* are electronic lists of data that have been optimized to perform a particular transaction. For example, Figure 11-1 shows a patient appointment file with information about patient's appointments, in the form in which it is used, so that the information can be accessed and processed quickly by the system.

A *database* is a collection of groupings of information that are related to each other in some way (e.g., through common fields). Logical groupings of information could include such categories as customer data, information about an order, and product information. A *database management system (DBMS)* is software that creates

| Appointment Date | Appointment Time | Duration | Reason | Patient ID | First Name | Last Name | Phone Number | Doctor ID | Doctor Last Name |
|------------------|------------------|----------|------------|------------|------------|-----------|--------------|------------|------------------|
| 11/23/2012 | 2:30 | .25 hour | Flu | 758843 | Patrick | Dennis | 548-9456 | V524625587 | Vroman |
| 11/23/2012 | 2:30 | 1 hour | Physical | 136136 | Adelaide | Kin | 548-7887 | T445756225 | Tantalo |
| 11/23/2012 | 2:45 | .25 hour | Shot | 544822 | Chris | Pullig | 525-5464 | V524625587 | Vroman |
| 11/23/2012 | 3:00 | 1 hour | Physical | 345344 | Felicia | Marston | 548-9333 | B544742245 | Brousseau |
| 11/23/2012 | 3:00 | .5 hour | Migraine | 236454 | Thomas | Bateman | 667-8955 | V524625587 | Vroman |
| 11/23/2012 | 3:30 | .5 hour | Muscular | 887777 | Ryan | Nelson | 525-4772 | V524625587 | Vroman |
| 11/23/2012 | 3:30 | .25 hour | Shot | 966233 | Peter | Todd | 667-2325 | T445756225 | Tantalo |
| 11/23/2012 | 3:45 | .75 hour | Muscular | 951657 | Mike | Morris | 663-8944 | T445756225 | Tantalo |
| 11/23/2012 | 4:00 | 1 hour | Physical | 223238 | Ellen | Whitener | 525-8874 | B544742245 | Brousseau |
| 11/23/2012 | 4:00 | .5 hour | Flu | 365548 | Jerry | Starsia | 548-9887 | V524625587 | Vroman |
| 11/23/2012 | 4:30 | 1 hour | Minor surg | 398633 | Susan | Perry | 525-6632 | V524625587 | Vroman |
| 11/23/2012 | 4:30 | .5 hour | Migraine | 222577 | Elizabeth | Gray | 667-8400 | T445756225 | Tantalo |
| 11/24/2012 | 8:30 | .25 hour | Shot | 858756 | Elias | Awad | 663-6364 | T445756225 | Tantalo |
| 11/24/2012 | 8:30 | 1 hour | Minor surg | 232158 | Andy | Ruppel | 525-9888 | V524625587 | Vroman |
| 11/24/2012 | 8:30 | .25 hour | Flu | 244875 | Rick | Grenci | 548-2114 | B544742245 | Brousseau |
| 11/24/2012 | 8:45 | .5 hour | Muscular | 655683 | Eric | Meier | 667-0254 | T445756225 | Tantalo |
| 11/24/2012 | 8:45 | 1 hour | Physical | 447521 | Jane | Pace | 548-0025 | B544742245 | Brousseau |
| 11/24/2012 | 9:30 | .5 hour | Flu | 554263 | Trey | Maxham | 663-8547 | V524625587 | Vroman |

FIGURE 11-1
Appointment File

and manipulates these databases (Figure 11-2). *End-user DBMSs* such as Microsoft Access support small-scale databases that are used to enhance personal productivity, and *enterprise DBMSs*, such as DB2, Jasmine, SQL Server, and Oracle, can manage huge volumes of data and support applications that run an entire company. An end-user DBMS is significantly less expensive and easier for novice users to use than its enterprise counterpart, but it does not have the features or capabilities that are necessary to support mission-critical or large-scale systems. Open-source DBMS's are also popular, such as MySQL.

The next section describes several different kinds of files and databases that can be used to handle a system's data storage requirements.

Files

A *data file* contains an electronic list of information that is formatted for a particular transaction, and the information is changed and manipulated by programs that are written for those purposes. Files created by older, legacy systems are frequently in a proprietary format, while newer systems use a standard format such as CSV (comma separated value) or tab-delimited. Typically, files are organized sequentially, and new records are added to the file's end. These records can be associated with other records by a pointer, which is information about the location of the related record. A pointer is placed at the end of each record, and it "points" to the next record in a series or set. Sometimes files are called *linked lists* because of the way the records are linked together by the pointers. There are several types of files that

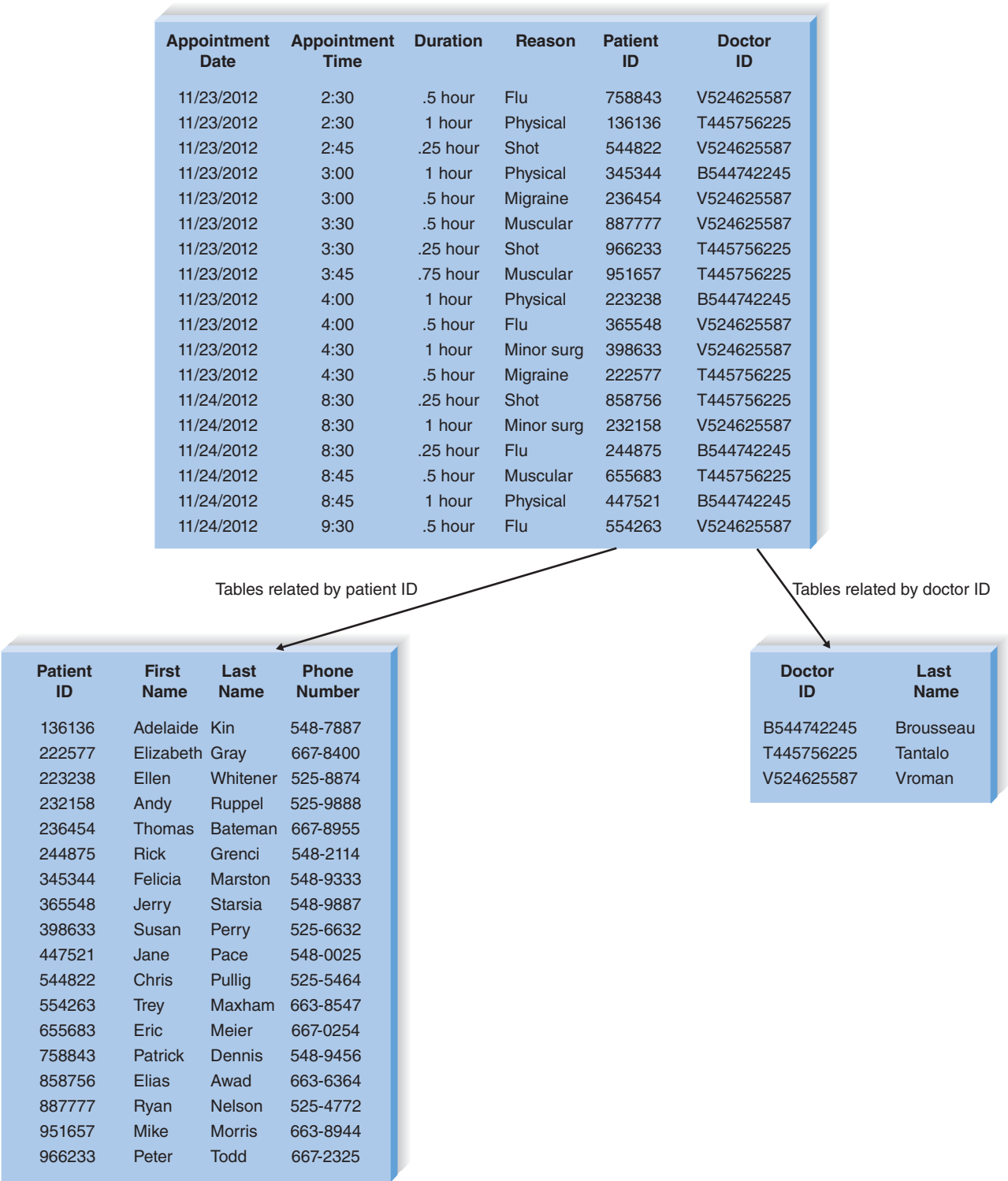


FIGURE 11-2
Appointment Database

differ in the way they are used to support an application: master files, look-up files, transaction files, audit files, and history files.

Master files store core information that is important to the business and, more specifically, to the application, such as order information or customer mailing information. They usually are kept for long periods, and new records are appended to the end of the file as new orders or new customers are captured by the system. If changes need to be made to existing records, programs must be written to update the old information.

Look-up files contain static values, such as a list of valid codes or the names of the U.S. states. Typically, the list is used for validation. For example, if a customer's mailing address is entered into a master file, the state name is validated against a look-up file that contains U.S. states to make sure that the value was entered correctly.

A *transaction file* holds information that can be used to update a master file. The transaction file can be destroyed after changes are added, or the file may be saved in case the transactions need to be accessed again in the future. Customer address changes, for one, would be stored in a transaction file until a program is run that updates the customer address master file with the new information.

For control purposes, a company might need to store information about how data changes over time. For example, as human resources clerks change employee salaries in a human resources system, the system should record the person who made the changes to the salary amount, the date, and the actual change that was made. An *audit file* records “before” and “after” images of data as the data are altered, so that an audit can be performed if the integrity of the data is questioned.

Sometimes files become so large that they are unwieldy, and much of the information in the file is no longer used. The *history file* (or archive file) stores past transactions (e.g., old customers, past orders) that are no longer needed by system users. Typically, the file is stored off-line, yet it can be accessed on an as-needed basis. Other files, such as master files, can then be streamlined to include only active or very recent information.

Databases

There are many different types of databases that exist on the market today. In this section, we provide a brief description of four databases with which you may come into contact: legacy, relational, object, and multidimensional. You will likely encounter a variety of ways to classify databases in your studies, but in this book we classify databases in terms of how they store and manipulate data.

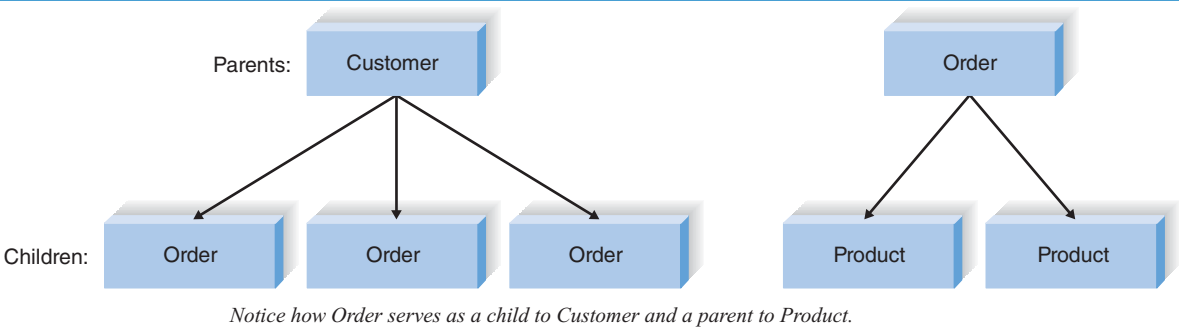
Legacy Databases The name *legacy database* is given to those databases which are based on older, sometimes outdated technology that is seldom used to develop new applications; however, you may come across them when maintaining or migrating from systems that already exist within your organization. Two examples of legacy databases include hierarchical databases and network databases. *Hierarchical databases* (e.g., IDMS) use hierarchies, or inverted trees, to represent relationships (similar to the one-to-many [1:M] relationships described in Chapter 6). The record at the top of the tree has zero or more child records, which in turn can serve as parents for other records (Figure 11-3). Hierarchical databases are known for rapid search capabilities and were used in the early systems in the airline industry.

YOUR
TURN

11-1 STUDENT ADMISSIONS SYSTEM

Pretend that you are building a Web-based system for the admissions office at your university. The system will be used to accept electronic applications from students. All the data for the system will be stored in a variety of files.

QUESTION: Give an example using the preceding system for each of the following file types: master, look-up, transaction, audit, and history. What kind of information would each file contain and how would the file be used?



Sample Records:

Customer as parent

| | |
|--------------|--------------|
| 1035 Black | ... |
| 235 | 11/23/11 ... |
| 1556 Fracken | ... |
| 236 | 11/23/11 ... |
| 243 | 11/26/11 ... |
| 2274 Goodin | ... |
| 237 | 11/23/11 ... |
| 245 | 11/26/11 ... |
| 260 | 11/30/11 ... |
| 275 | 12/7/11 ... |
| 4254 Bailey | ... |
| 234 | 11/23/11 ... |
| 242 | 11/26/11 ... |
| 9500 Chin | ... |
| 233 | 11/23/11 ... |
| 244 | 11/26/11 ... |
| 262 | 11/30/11 ... |

Order as parent

| | | | |
|-----|---------------|-----|----------------|
| 233 | 11/23/11 ... | 444 | Wine Gift Pack |
| 222 | Bottle Opener | 555 | Cheese Tray |
| 234 | 11/23/11 ... | 222 | Bottle Opener |
| 235 | 11/23/11 ... | 555 | Cheese Tray |
| 222 | Bottle Opener | 333 | Jams & Jellies |
| 236 | 11/23/11 ... | 222 | Bottle Opener |
| 237 | 11/23/11 ... | 111 | Wine Guide |
| 242 | 11/26/11 ... | 444 | Wine Gift Pack |
| 243 | 11/26/11 ... | 333 | Jams & Jellies |
| 222 | Bottle Opener | 555 | Cheese Tray |

FIGURE 11-3
Hierarchical Database

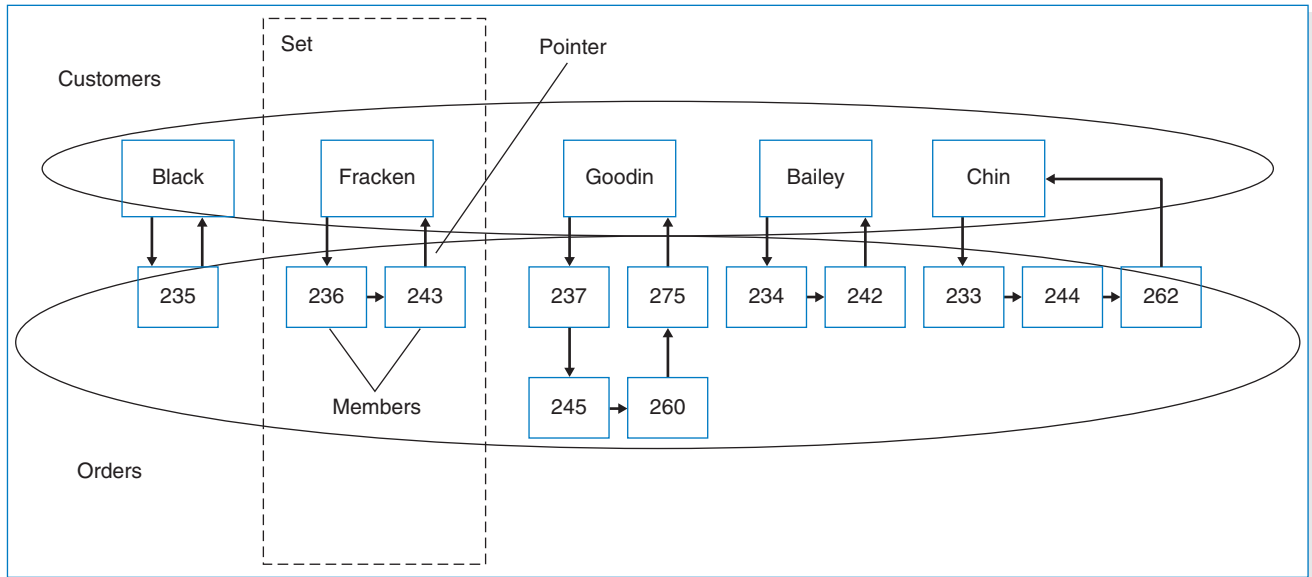


FIGURE 11-4
Network Database

Hierarchical databases cannot efficiently represent many-to-many (M:N) relationships or nonhierarchical associations—a major drawback—so network databases were developed to address this limitation (and others) of hierarchical technology. *Network databases* (e.g., IDMS/R, DBMS 10) are collections of records that are related to each other through *pointers*. Basically, a record is a *member* in one or more *sets*, and the pointers link the members in a set together (Figure 11-4).

Both kinds of legacy systems can handle data quite efficiently, but they require a great deal of programming effort. The application system software needs to contain code that manipulates the database pointers; in other words, the application program must understand how the database is built and be written to follow the structure of the database. When the database structure is changed, the application program must be rewritten to change the way it works, which makes the application using the databases difficult to build and maintain. The code required to maintain the pointers can be quite error prone, especially if bidirectional pointers were used. Years ago, when hardware was expensive and programmer time was cheap, hierarchical and network databases were good solutions for large systems; however, as hardware costs dropped and people costs skyrocketed, these solutions became much less cost effective.

Relational Databases *The relational database* is the most popular kind of database for application development today. Although it is less “machine efficient” than its legacy counterparts, it is much easier to work with from a development perspective. A relational database is based on collections of tables, each of which has a *primary key*—a field(s) whose value is different for every row of the table. The tables are related to each other by the placement of the primary key from one table into the related table as a *foreign key* (Figure 11-5).

Most relational database management systems (RDBMSs) support *referential integrity*, or the idea of ensuring that values linking the tables together through the

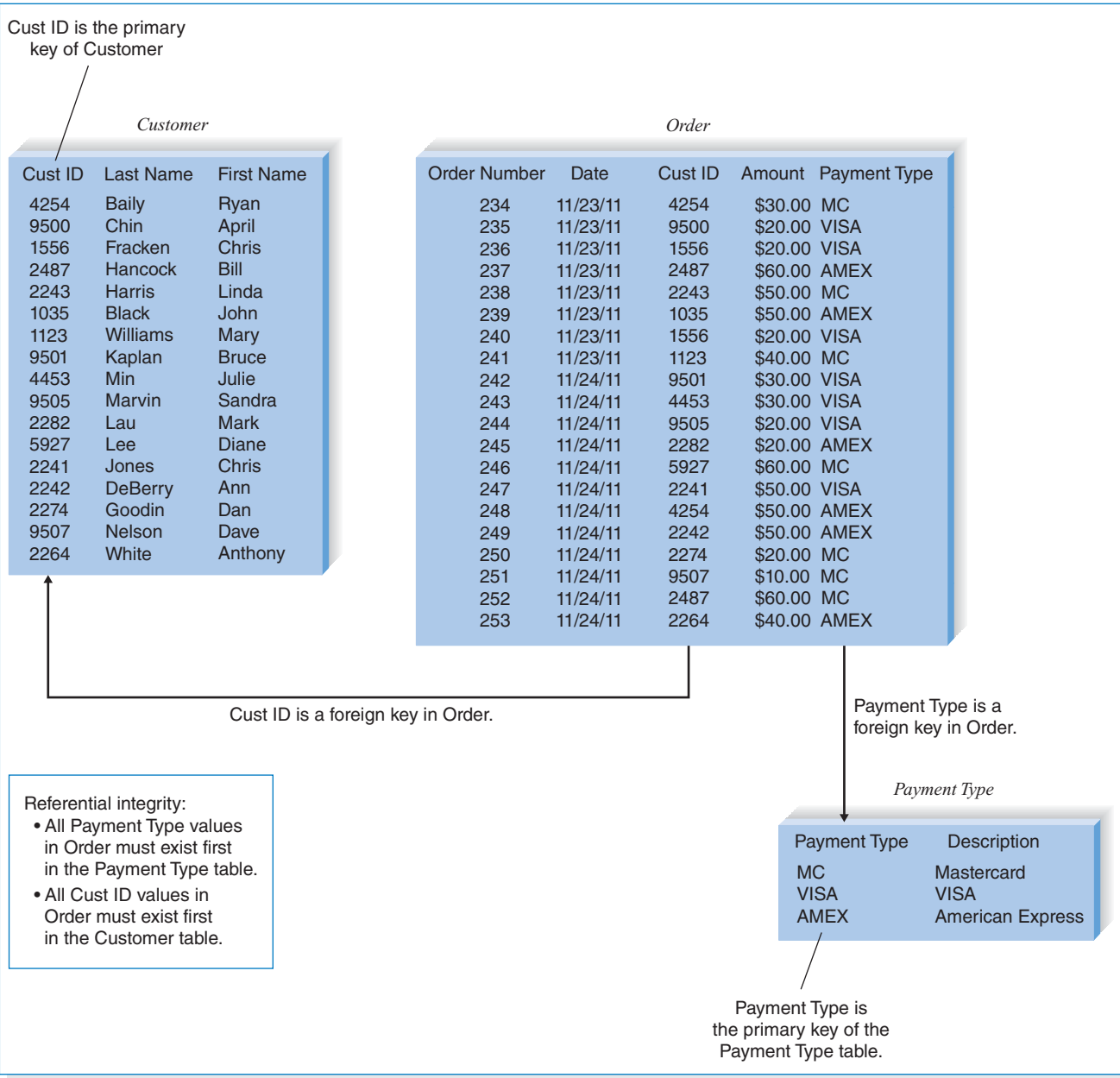


FIGURE 11-5
Relational Database

primary and foreign keys are valid and correctly synchronized. For example, if an order-entry clerk using the tables in Figure 11-5 attempted to add order 254 for customer number 1111, he or she would have made a mistake because no customer exists in the Customer table with that number. If the RDBMS supported referential integrity, it would check the customer numbers in the Customer table, discover that the number 1111 is invalid, and return an error to the entry clerk. The clerk would then go back to the original order form and recheck the customer information. Can you imagine the problems that would occur if the RDBMS let the entry clerk add

the order with the wrong information? There would be no way to track down the name of the customer for order 254.

Tables have a set number of columns and a variable number of rows that contain occurrences of data. *Structured Query Language (SQL)* is the standard language for accessing the data in the tables, and it operates on complete tables, as opposed to the individual records in the tables. Thus, a query written in SQL is applied to all the records in a table all at once, which is different from a lot of programming languages that manipulate data record by record. When queries must include information from more than one table, the tables first are joined together on the basis of their primary key and foreign key relationships and treated as if they were one large table. Examples of RDBMS software are Microsoft Access, Oracle, DB2, Sybase, Informix, Microsoft SQL Server, and MySQL.

Object Databases The next type of database is the *object database*, or object-oriented database. (See Chapter 14 for more information on object-oriented approaches.) The basic premise of object orientation is that all things should be treated as objects that have both data (attributes) and processes (behaviors). An object changes or accesses its own attributes only through its behaviors. Objects may communicate with each other for information or certain actions. Changes to one object have no effect on other objects because the attributes and behaviors are self-contained, or encapsulated, within each one. This *encapsulation* allows objects to be reused to build many different systems, because they can be inserted and removed from applications with few ripple effects. For example, a customer object could be defined one time as having attributes (e.g., customer number, customer name) and behaviors (e.g., inserting a customer, deleting a customer), and then this customer object could be used to build any system that involves a customer.

In object databases, the combination of data and processes is represented by *object classes*, which are the major categories of objects in the system, and a class can contain a variety of *subclasses*, or special cases of that class. For example, a person class can have subclasses of employee and customer because employee and customer are special cases of person. An instance of data in object databases is referred to as an *instantiation* (e.g., *John Smith* is an instantiation of the *customer* object), and the relationships among classes are maintained by pointers (Figure 11-6).

Object-oriented database management systems (OODBMSs) are mainly used to support multimedia applications or systems that involve complex data (e.g., graphics, video, and sound). Telecommunications, financial services, health care, and transportation have been the most receptive to object databases. They are becoming a popular technology for supporting electronic commerce, online catalogs, and large Web multimedia applications.

Although pure OODBMSs like Jasmine exist, most organizations invest in *hybrid OODBMS* technology, which includes databases with both object and relational features. For instance, Oracle, a leader in the relational database market, incorporates object functionality and capabilities into its relational product.

Although the market for OODBMSs is expected to grow, the market for the technology is dwarfed by that for its relational and object-relational database counterparts (\$13.8 billion).¹ For one, there are many more experienced developers and tools in the relational database arena. Also, relational users find that OODBMS technology comes with a very steep learning curve.

¹ Barbara Darrow, "Linux, SQL Server Drive Database Market: Report," *ChannelWeb*, May 24, 2006.

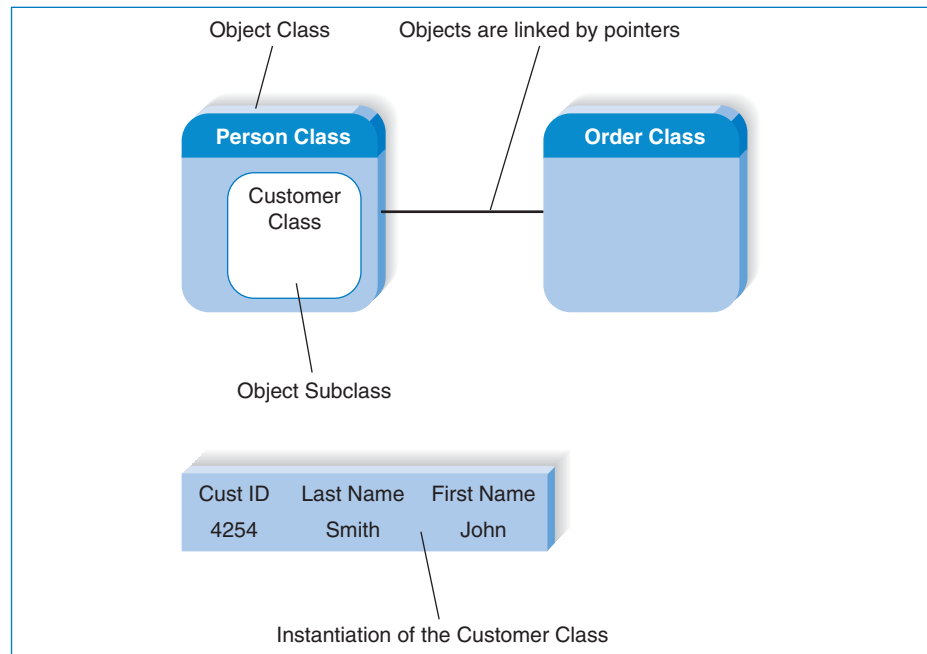


FIGURE 11-6
Object Database

Multidimensional Databases A multidimensional database is a type of relational database that is used extensively in data warehousing. *Data warehousing* is the practice of taking data from a company's transaction processing systems, transforming the data (e.g., cleaning them up, totaling them, aggregating them), and then storing the data for use in a data warehouse (i.e., a large database) that supports *decision support systems (DSS)*. A data warehouse itself usually relies on relational technology as its storage format; however, companies can create *data marts*, which are smaller databases based on data warehouse data. Typically, a data mart receives downloads of data from the data warehouse regularly, and it supports DSS for a specific department or functional area of the company. For example, the marketing department may have a data mart that supports its campaign management DSS. Data marts are usually created with multidimensional databases.

In most cases, DSS is designed not to search for a particular record (e.g., "What did John Smith order on July 5, 2011?"), but rather to display information that is *aggregated* (e.g., totaled or averaged) across many records (e.g., "What was the average sales by quarter for product A?") Thus, data marts that support a DSS require that data be stored in a format in which they can be easily aggregated and manipulated across a variety of dimensions (e.g., time, product, region, sales rep). Unfortunately, legacy, object, and relational databases are designed and optimized to provide access to individual records, not to store data to support aggregations of data on multiple dimensions.

When data are first loaded into a multidimensional database, the database precalculates the data across multiple dimensions and stores the answers, using arrays or some other technique. Although the initial loading of the data can be quite slow because of all the calculations that must take place, data access is extremely fast because the "answers" already exist in the arrays. For example, the cube in Figure 11-7 represents a multidimensional database containing data that

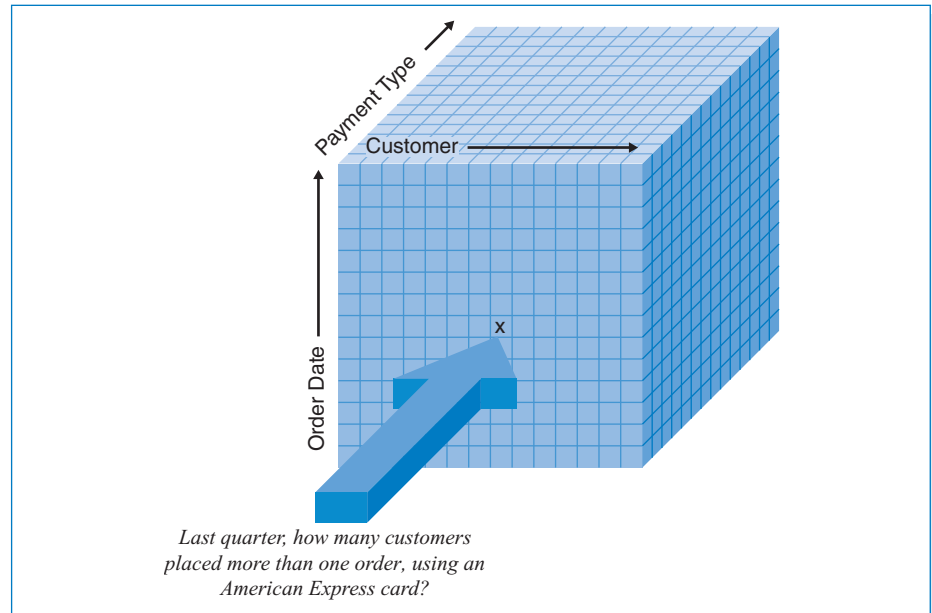


FIGURE 11-7
Multidimensional Database

have been organized by customer, payment type, and order date. Precalculated quantitative information (e.g., totals, averages) is stored at the intersection of the dimensions (in each block), and the DSS directly accesses those blocks. Because blocks contain precalculated information, there is much less processing that needs to occur to provide the DSS with aggregated results.

Selecting a Storage Format

Each of the file and database data storage formats has its strengths and weaknesses, and no one format is inherently better than the others. In fact, sometimes, a project team will choose multiple data storage formats (e.g., a relational database for one data store, a file for another, and a multidimensional database for a third). Thus, it is important to understand the strengths and weaknesses of each format and when to use each one. Figure 11-8 summarizes the characteristics of each and the characteristics that can help identify when each type of storage is most appropriate.

Data Types The first issue is the type of data that will need to be stored in the system. Most applications need to store simple data types, such as text, dates, and numbers, and all DBMSs are equipped to handle this kind of data. The best choice for simple data storage, however, usually is the relational database because the technology has matured over time and has continuously improved to handle simple data very effectively.

Increasingly, applications are incorporating complex data, such as video, images, or audio, and object databases are best able to handle data of this type. Complex data are stored as objects that can be manipulated much faster than with other storage formats. Other applications require aggregated data (i.e., information that has been summed, averaged, or combined in some way). Multidimensional databases are specially designed to store data so that they can be “sliced and diced”



| | Files | Legacy DBMS | Relational DBMS | Object-Oriented DBMS | Multi-dimensional DBMS |
|--|---|---|--|--|---|
| Major strengths | Files can be designed for fast performance; good for short-term data storage. | Very mature products | Leader in the database market; can handle diverse data needs | Able to handle complex data | Configured to answer decision support questions quickly |
| Major weaknesses | Redundant data; data must be updated, using programs. | Not able to store data as efficiently; limited future | Cannot handle complex data | Technology is still maturing; skills are hard to find. | Highly specialized use; skills are hard to find |
| Data types supported | Simple | <i>Not recommended for new systems</i> | Simple | Complex (e.g., video, audio, images) | Aggregated |
| Types of application systems supported | Transaction processing | <i>Not recommended for new systems</i> | Transaction processing and decision making | Transaction processing | Decision making |
| Existing data formats | Organization dependent | Organization dependent | Organization dependent | Organization dependent | Organization dependent |
| Future needs | Limited future prospects | Poor future prospects | Good future prospects | Uncertain future prospects | Uncertain future prospects |

DBMS = database management system.

FIGURE 11-8
Comparison of Data Storage Formats

and examined across important business dimensions. If the system is being built for analytical decision support, then this option likely will be most appropriate.

Type of Application System There are many different kinds of application systems that can be developed. *Transaction processing systems* are designed to accept and process many simultaneous requests (e.g., order entry, distribution, payroll). In transaction processing systems, the data are continuously updated by a large number of users, and the queries that are asked of the systems typically are predefined or targeted at a small subset of records (e.g., “List the orders that were back-ordered today”; “What products did customer #1234 order on May 12, 2011?”).

Another set of application systems comprises those designed to support decision making, such as business intelligence *management information systems* (MISs), *executive information systems* (EISs), and *expert systems* (ESs). These decision support systems (DSS) are built to support decision makers who need to examine large amounts of read-only historical data. The questions that they ask often are ad hoc, and they include hundreds or thousands of records at a time (e.g., “List all customers in the west region who purchased a product costing more than \$500 at least three times”; “What products had increased sales in the summer months but have not been classified as summer merchandise?”).

Transaction processing and DSSs thus have very different data storage needs. Transaction processing systems need data storage formats that are tuned for a lot of data updates and fast retrieval of predefined, specific questions. Files, relational databases, and object databases can all support these kinds of requirements.

By contrast, systems to support decision making are usually only reading data (not updating it), often in ad hoc ways. The best choices for these systems usually are relational databases and multidimensional databases because these formats can be configured specially for needs that may be unclear and less apt to change the data.

Existing Storage Formats The data storage format should be selected primarily on the basis of the kind of data and application system being developed. Project teams should also consider the existing data storage formats in the organization when making design decisions. In this way, they can better understand the technical skills that already exist and how steep the learning curve will be when the data storage format is adopted. For example, a company that is familiar with relational databases will have little problem adopting a relational database for the project, whereas an object database may require substantial developer training. In the latter situation, the project team may have to plan for more time to integrate the object database with the company's relational systems.

Future Needs The project team should be aware of current trends and technologies that are being used by other organizations. A large number of installations of a type of data storage format suggests that the selection of that format is “safe,” in that needed skills and products are available. For example, it would probably be easier and less expensive to find relational database expertise when implementing a system than to find help with a multidimensional data storage format. Legacy database skills, too, would likely be difficult to find.

Applying the Concepts at Tune Source

The Tune Source Digital Music Download system needs to effectively present tune information to users and capture purchase data. Jason Wells, senior systems analyst and project manager for the Digital Music Download system, recognized that these goals were dependent on a good design of the data storage component for the new application.

The project team met to discuss two issues that would drive the data storage format selection: what kind of data would be in the system and how that data would be used by the application system. Using a white board, they listed the ideas presented in Figure 11-9. The project team agreed that the bulk of the data in the system would be text and numbers describing customers and purchases that are exchanged with Web users. A relational database would be able to handle the data effectively, and the technology would be well received because of its current use at Tune Source.

The team recognized, however, that relational technology may not be optimized to handle complex data such as the images, sound clips, and video clips associated with the application. Jason asked Kenji, a project team member, to investigate relational databases that offered object add-on products. It might be possible to invest in a relational database foundation and use its object functionality to handle the complex data.

The team noted that one transaction file must be designed to handle the interface with the Web shopping cart program. The team must design the file that stores temporary, purchase information on the Web server as customers navigate through the Web site.

| Data | Type | Use | Suggested Format |
|--------------------------------|---|--------------|------------------|
| Customer information | Simple (mostly text) | Transactions | Relational |
| Sales information | Simple (text and numbers) | Transactions | Relational |
| Tune information | Both simple and complex (the system will contain audio clips, video, etc.) | Transactions | Relational ? |
| Interests/Favorites | Simple (mostly text) | Transactions | Relational |
| Targeted promotion information | Simple text, formatted specifically for populating the Web site with customized content | Transactions | Relational |
| Temporary information | The system will likely need to hold information for temporary periods (e.g., the shopping cart will store purchase information before the purchase is actually completed) | Transactions | Transaction file |

FIGURE 11-9
Types of Data in the Digital Music Download System

Of course, Jason realized that other data needs would arise over time, but he felt confident that the major data issues were identified (e.g., the ability to handle complex data) and that the data storage design would be selected on the basis of the proper storage technologies.

MOVING FROM LOGICAL TO PHYSICAL DATA MODELS

During analysis, the analysts defined the data required by the application by creating *logical entity relationship diagrams (ERDs)*. These logical models depict the “business view” of the data, but omit any implementation details. Now, having determined the data storage format, *physical data models* are created to show implementation details and to explain more about the “how” of the final system. These to-be models describe characteristics of the system that will be created, communicating the “systems view” of the new system.

The Physical Entity Relationship Diagram

Like the DFD, the ERD contains the same components for both the logical and physical models, including entities, relationships, and attributes. The difference lies in the fact that *physical ERDs* contain references to exactly how data will be stored in a file or database table and that much more metadata is added to the CASE repository to describe the data model components. The transition from the logical to physical data model is fairly straightforward; see the steps in Figure 11-10.

Step 1: Change Entities to Tables or Files The first step is to change all the entities in the logical ERD to reflect the files or tables that will be used to store the data.

| Step | Explanation |
|-------------------------------------|---|
| Change entities to tables or files. | Beginning with the logical entity relationship diagram, change the entities to tables or files and update the metadata. |
| Change attributes to fields. | Convert the attributes to fields and update the metadata. |
| Add primary keys. | Assign primary keys to all entities. |
| Add foreign keys. | Add foreign keys to represent the relationships among entities. |
| Add system-related components. | Add system-related tables and fields. |

FIGURE 11-10
Steps to Moving from Logical to Physical
Entity Relationship Diagram

Usually, project teams adhere to strict naming conventions for such things as tables, files, and fields, so the physical ERD would use the names that the real components will have when implemented. Metadata for the tables and files, like the expected size of the table, are added to the CASE repository. See Figure 11-11 for a physical ERD from the Lawn Chemical Request system that was described in Chapters 5 and 6.

Step 2: Change Attributes to Fields Second, change the attributes to fields, which are columns in files or tables, and add information like the field’s length, data type, default value, and valid value to the CASE repository. There are a number of different data types that fields can have, such as number, decimal, longint, character, and variable character. The analyst inputs the data type along with the size of the field into the CASE tool so that the system can be designed for the right kind of information. A *default value* specifies what should be placed in a column if no value is explicitly supplied when a record is inserted into the table. A *valid value* is a fixed list of valid values for a particular column, or an expression to define some form of data validation code for a column or table. Figure 11-12 shows a variety of metadata describing the cust_id field in an Oracle Customer table.

Inputting complete information regarding the tables and columns into the CASE repository is very important. Many CASE tools will actually generate code to build tables and create files for the new system according to the information they contain for the physical models. By taking time to describe the physical data model in detail, the analyst can save a lot of time when the system is ready to be implemented.

Step 3: Add Primary Keys As a third step, the attributes that served as identifiers on the logical ERD are converted into *primary keys*, which are fields that contain a

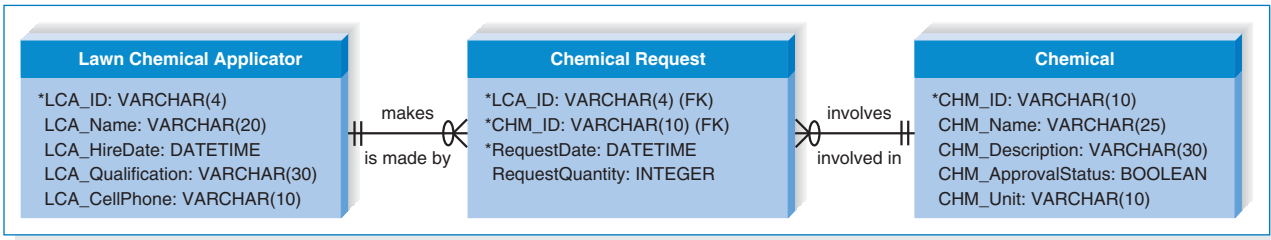


FIGURE 11-11
Lawn Chemical Request System Physical ERD

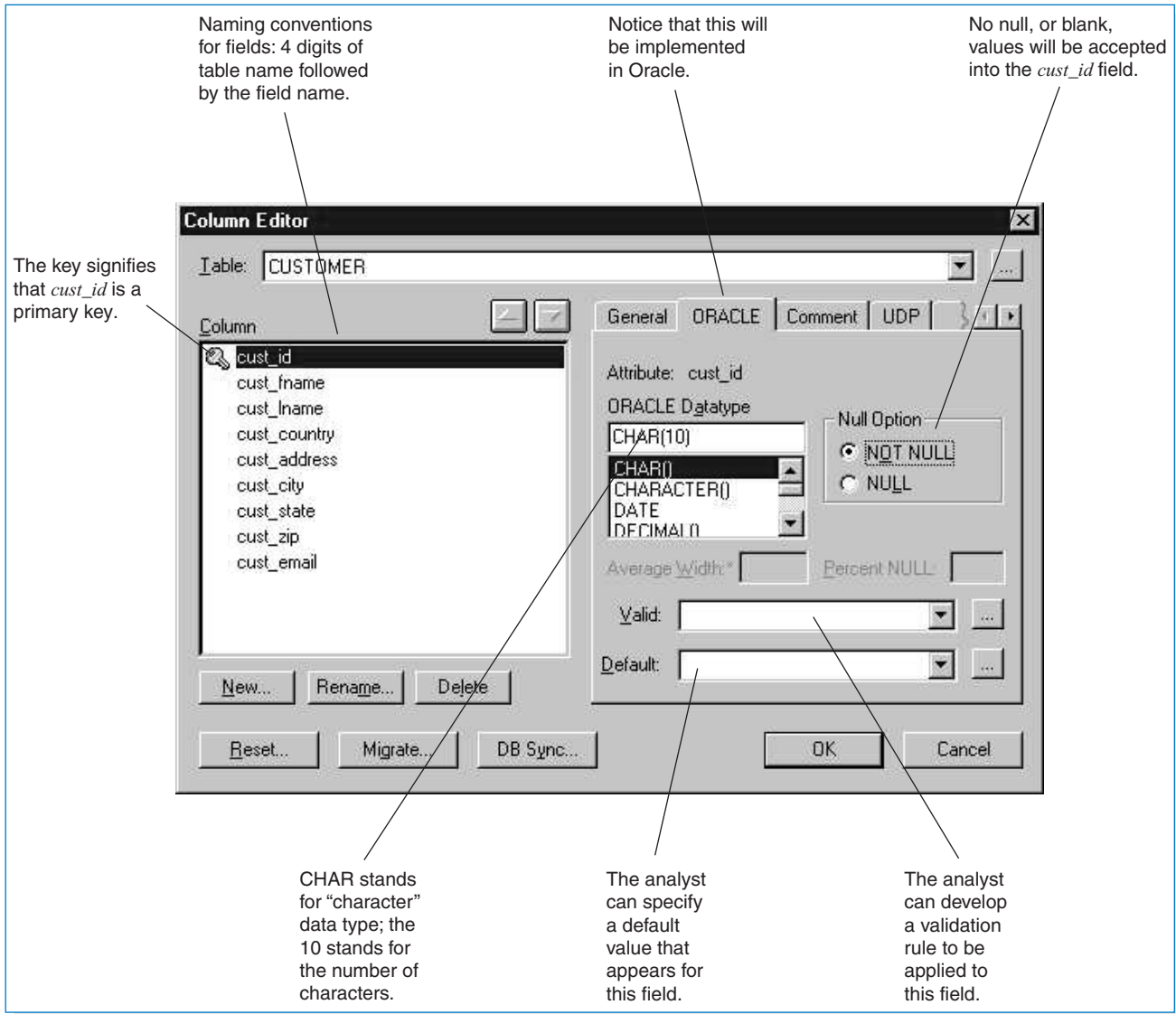


FIGURE 11-12
Metadata for a *cust_id* Field

unique value for each record in the file or table. For instance, Social Security Number would serve as a good primary key for a customer table if every customer record in the table will contain a unique value in the Social Security Number field. A unique identifier is mandatory for every table placed on the physical ERD; therefore, primary key fields must be created for entities that did not have identifiers previously. For example, if we did not choose an identifier for the customer entity on the logical ERD, we would now create a system-generated field (e.g., *cust_id*) that could serve as the primary key for the customer table. This field would have no meaning or purpose other than ensuring that each record has a field that contains a unique value.

Step 4: Add Foreign Keys The relationships on the logical ERD show that pairs of entities are associated with each other, and in step 4, the analyst specifies how

the associations are going to be maintained from a technical standpoint. In a relational database, for example, an association between two tables is maintained by a technique referred to as a *foreign key*. A foreign key is the primary key field(s) from one table that is repeated in another table to provide a common field between the two tables. The common field contains values that match a record in one table to a record in the other. For example, if we were to create two tables called Customer and Order that were related to each other, we could include the primary key field from Customer (cust_id) in the Order table as well. In this way, if we want to find out customer information (e.g., name, address, phone number) when looking at someone's order, we can use the value for cust_id that appears in the Order table to go back to the Customer table to locate the appropriate information.

Thus, on the physical ERD, the primary key fields in the parent tables (the “1” end of the relationship) are copied and placed as fields in the child tables (the “many” end of the relationship) and designated as foreign keys. The fields will contain values that are common between the two tables. Many times, the CASE tools that are used to draw ERDs will “migrate” foreign keys to the appropriate tables on the model automatically, and the database technology will ensure that the values in the two fields match appropriately, helping to ensure referential integrity.

Step 5: Add System-Related Components As the fifth and final step, components are added to the physical ERD to reflect special implementation needs, including components that were included on the DFD. We have mentioned balance between DFDs and ERDs in earlier chapters, and this balance must be maintained in the physical models as well. Therefore, implementation-specific data stores and data elements from the physical DFD should be included on the ERD as tables and fields. For example, in Figure 10-2 we added the Tune to buy history data store to the physical DFD to serve as a “backup” for tunes that are sent to the purchase tunes process. Now we will need to add a tune to buy batch history file to the physical ERD model along with its fields and relationships.

Revisiting the CRUD Matrix

As discussed in Chapter 6, it is important to verify that the system's DFD and ERD models are balanced. In other words, we must ensure that data needed in the systems processes are stored and that all stored data are used by at least one process. The CRUD matrix was introduced in Chapter 6 as a tool showing how data are used by processes in the system.

Often the CRUD matrix is created during analysis on the basis of the logical process and data models. In design, as these models are converted to physical models, changes in the form of new processes, new data stores, and new data elements may occur. The CRUD matrix should be revised at this point to include the new components and ensure that balance is maintained between the physical ERD and DFDs.

If the CRUD matrix was not developed during analysis, it should be developed now prior to implementation. The matrix shows exactly how data are used and created by the major processes in the system, so it serves as a very useful component of the system design materials.

Applying the Concepts at Tune Source

Let us now apply some of the concepts that you have learned by creating a physical ERD, using the logical ERD that was created in Chapter 6.

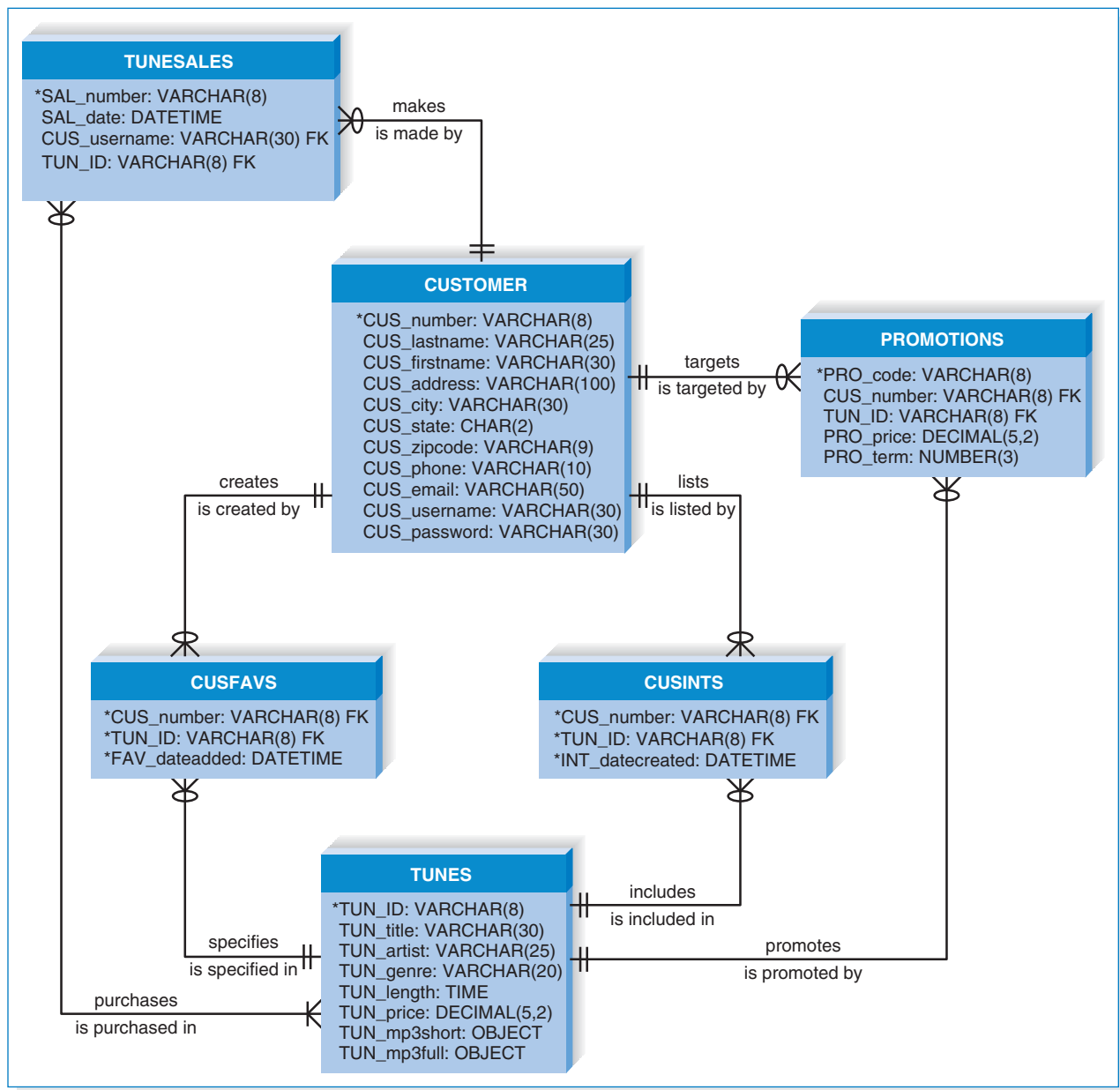


FIGURE 11-13
Tune Source Physical ERD

When we use the logical model as a starting point, the first step is to rename the entities to match with the tables or files that will be used by the system (Figure 11-13). Outwardly, the data model does not look very different after this step, but notice that several entities have been renamed to be consistent with Tune Source’s table naming standards. At this time, we will need to include metadata for the tables, such as their estimated size.

Next, the attributes for the entities become fields with such characteristics as data type, length, and valid values, and this is recorded in the CASE repository. For example, CUS_state in the CUSTOMER table will be a text field with a size of two

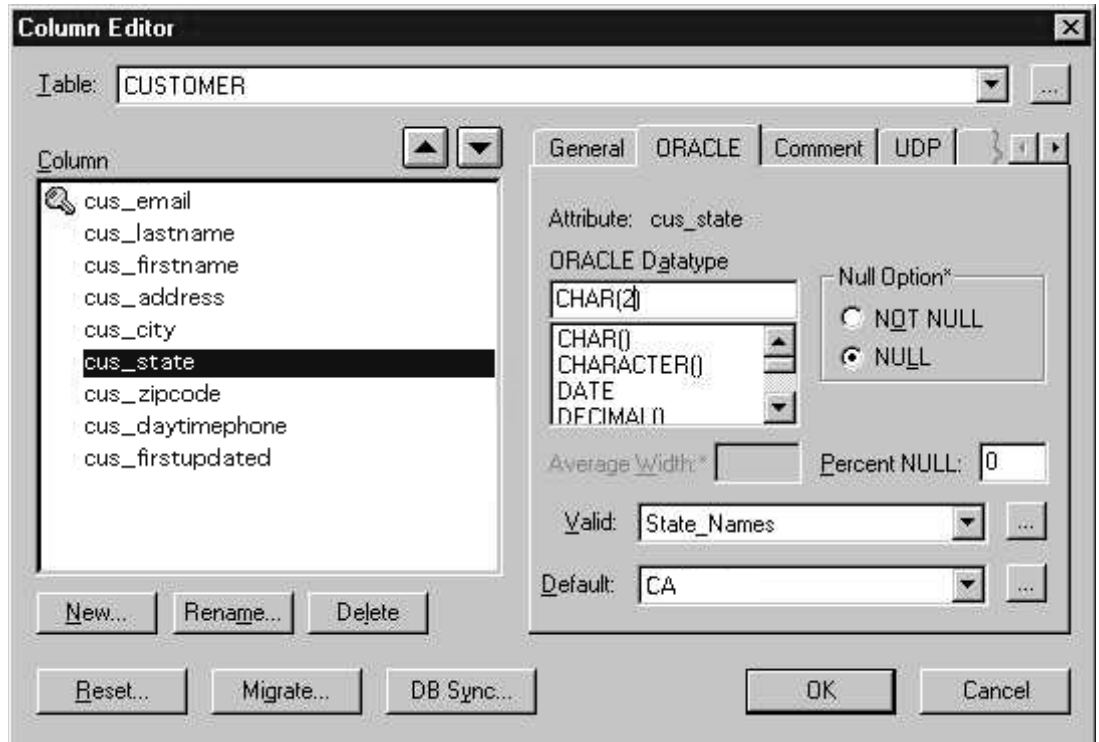


FIGURE 11-14
Computer-Aided Software Engineering Repository Entry for `cus_state` Field

characters, and valid values are the 50 two-letter state abbreviations. If most customers at Tune Source live in the state of California, then it may be worthwhile to make CA the default value for this field. However, since this is an Internet-based system, this assumption may not be valid. Figure 11-14 is an example of the CASE repository entry for the `CUS_state` field.

Step 3 suggests that we change the identifiers in the logical ERD to become primary keys, and entities without identifiers need to have a primary key created. At this time, we also can decide to use a system-generated primary key if it is more efficient than using logical attributes from the logical model.

The relationships on the logical ERD indicate where foreign key fields need to be placed. For example, `CUS_number` is placed as a field in `TUNESALES` to serve as the link between two entities, and `TUNESALES` gets the extra field because it is the child table (it exists at the “many” end of the relationship). Similarly, `TUN_ID` is placed in the `TUNESALES` table.

Finally, system-related components are included within the model. For example, fields that will capture when a record was last inserted or updated were added to many of the tables.

The project team also updated the CRUD matrix for the system. Figure 11-15 shows the CRUD matrix that was created for the Tune Source search and browse tunes process. Look at the original process models, and notice how the first process is merely reading information from data stores. This is illustrated on the CRUD matrix by an “R” placed in the relevant intersections of the matrix. Can you tell how data are used by the remaining processes?

| | 1.1 Load Web Site | 1.2 Process Search Requests | 1.3 Process Tune Selection |
|-----------------|-------------------|-----------------------------|----------------------------|
| PROMOTIONS | | | |
| PRO_code | R | | |
| CUS_number | R | | |
| TUN_ID | R | | |
| PRO_price | R | | |
| PRO_term | R | | |
| CUSFAVS | | | |
| CUS_number | R | | C |
| TUN_ID | R | | C |
| FAV_dateadded | R | | C |
| TUNES | | | |
| TUN_ID | | R | R |
| TUN_title | | R | R |
| TUN_artist | | R | R |
| TUN_genre | | R | R |
| TUN_length | | R | R |
| TUN_price | | R | R |
| TUN_mp3short | | R | R |
| TUN_mp3full | | R | R |
| CUSINTS | | | |
| CUS_number | | | C |
| TUN_ID | | | C |
| INT_datecreated | | | C |

FIGURE 11-15
CRUD Matrix for Search and Browse
Tunes Process

OPTIMIZING DATA STORAGE

The selected data storage format is now optimized for processing efficiency. The optimization methods will vary with the format that you select; however, the basic concepts will remain the same. Once you understand how to optimize a particular type of data storage, you will have some idea as to how to approach the optimization of other formats. This section focuses on the optimization of the most popular data storage format: relational databases.

There are two primary dimensions in which to optimize a relational database: for storage efficiency and for speed of access. Unfortunately, these two goals often conflict because the best design for access speed may take up a great deal of storage space as compared with other less speedy designs. This section describes how to use normalization (Chapter 6) to optimize data storage for storage efficiency. The next section presents design techniques, such as denormalization and indexing, that will quicken the performance of the system. Ultimately, the project team will go through a series of trade-offs until the ideal balance between both optimization dimensions is reached. Finally, the project team must estimate the size of the data storage needed to ensure that there is enough capacity on the server(s).

YOUR

TURN

11-2 ISLAND CHARTERS

In Chapter 6, you were asked to create a logical entity relationship diagram (ERD) for a charter company that owns boats that are used to charter trips to the islands ("Your Turn 6-8"). The company has created a computer system to track the boats it owns, including each boat's ID number, name, and seating capacity. The company also tracks information about the various islands, such as name and population. Every time a boat is chartered, it is important to know the data about the

trip that takes place and the number of people on the trip. The company also keeps information about each captain, such as Social Security Number, name, birthdate, and how to contact next of kin. Boats travel to only one island per visit.

Create a physical ERD for this situation. Compare the diagram that you drew to the logical diagram that you created in Chapter 6.

Optimizing Storage Efficiency

The most efficient tables in a relational database in terms of storage space have no redundant data and very few null values, because the presence of these suggest that space is being wasted (and more data to store means higher data storage hardware costs). For example, notice that the sample order table in Figure 11-16 repeats customer information, such as name and state, each time a customer places an order, and it contains many null values in the last four columns. These null values occur whenever a customer places an order for less than three items (the maximum number on an order).

In addition to wasting space, redundancy and null values also allow more room for error and increase the likelihood that problems will arise with the integrity of the data. What if customer 1135 moves from Maryland to Georgia? In the case of Figure 11-16, a program must be written to ensure that all instances of that customer are updated to show "GA" as the new state of residence. If some of the instances are overlooked, then the table will contain an update anomaly whereby some of the records contain the correctly updated value for state and other records contain the old information.

YOUR

TURN

11-3 DONATION TRACKING SYSTEM

A major public university graduates approximately 10,000 students per year, and its development office has decided to build a Web-based system that solicits and tracks donations from the university's large alumni body. Ultimately, the development officers hope to use the information in the system to better understand the alumni giving patterns so that they can improve giving rates.

QUESTION:

1. What kind of system is this?
2. Does it have characteristics of more than one?
3. What different kinds of data will this system use?
4. On the basis of your answers, what kind of data storage format(s) do you recommend for this system?

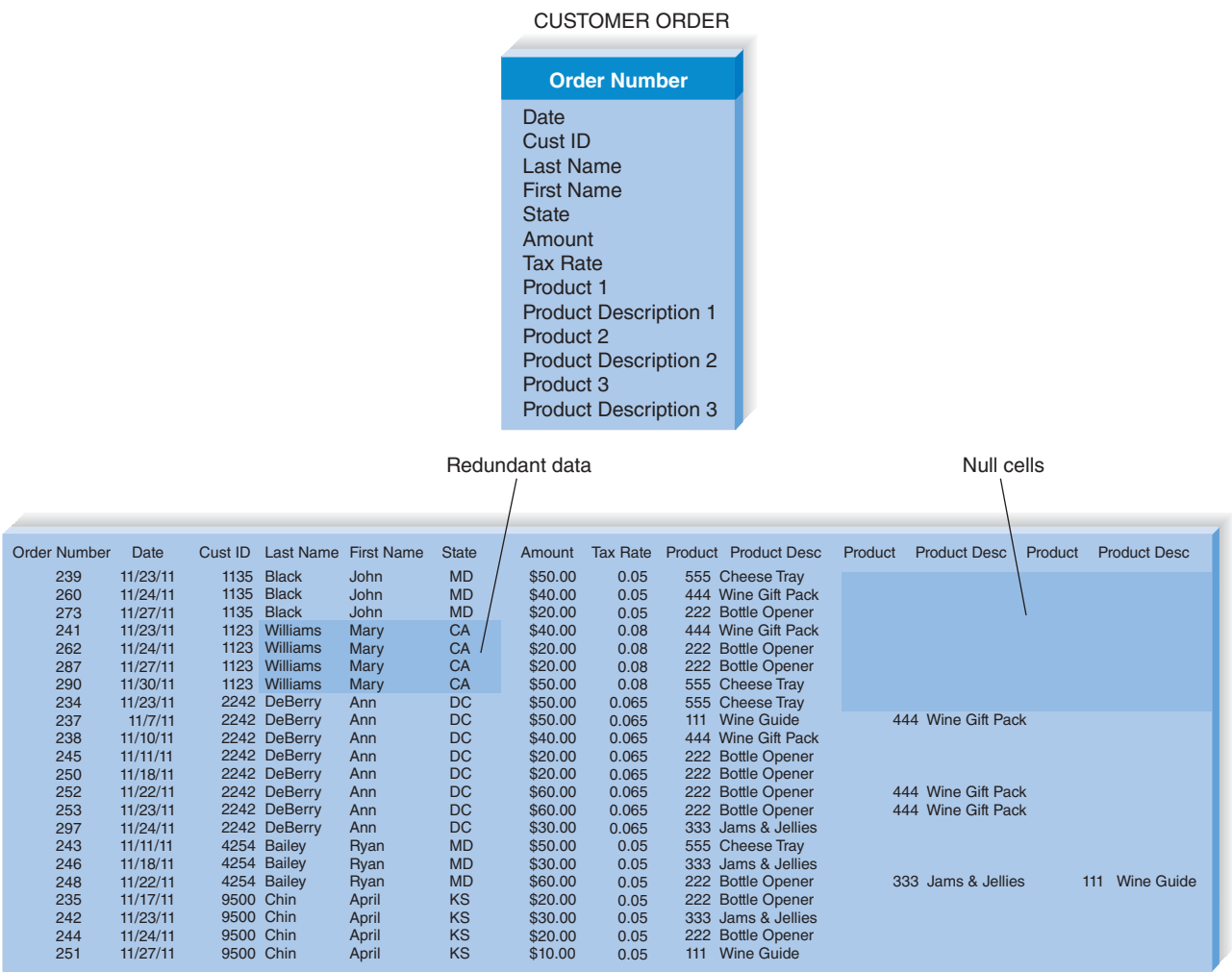


FIGURE 11-16
Optimizing Data Storage

Null values threaten data integrity because they are difficult to interpret. A blank value in the customer order table’s product fields could mean that (1) the customer did not want more than one or two products on his or her order, (2) the operator forgot to enter in all three products on the order, or (3) the customer canceled part of the order and the products were deleted by the operator. It is impossible to be sure of the actual meaning of the null values.

For both of these reasons—wasted storage space and data integrity threats—project teams should remove redundancy and null values from data storage design. During the design phase, the logical data model is used to examine the data storage design and optimize it for storage efficiency. If you follow the modeling instructions and guidelines that were presented in Chapter 6, you will have little trouble creating a design that is highly optimized in this way, because a well-formed logical data model does not contain redundancy or many null values.

Sometimes, however, a project team starts with a logical model that was poorly constructed or with a model that was created for files or a nonrelational type

of data storage format. In these cases, the project team should follow the steps of the *normalization* process described in Chapter 6 (see Figure 6A-1). Normalization is the best way to optimize data storage for efficiency.

Optimizing Access Speed

After you have optimized your data model design for data storage efficiency, the end result is data that are spread out across a number of tables. When data from multiple tables must be accessed or queried, the tables first must be joined together. For example, in Figure 11-2, before the office manager can print out a list of appointments with patient and doctor names on it, the patient and doctor tables need to be joined with the appointment table on the basis of the patient ID and doctor ID fields. Only then can appointment, patient, and doctor information be included in the query's output. Joins can take a lot of time, especially if the tables are large or if many tables are involved.

Consider an order system that stores information about 10,000 different products, 25,000 customers, and 100,000 orders, each order containing three products, on average. If an analyst wanted to investigate whether there were regional differences in buying preferences, he or she would need to combine all of the tables to be able to look at products that have been ordered, while knowing the location of the customers placing the orders. A query of this information would result in a huge table with 300,000 rows (i.e., the number of products that have been ordered) and many columns representing columns from all three tables combined.

There are several techniques that the project team can use to try to speed up access to the data: denormalization, clustering, indexing, and estimating the size of the data for hardware planning purposes.

Denormalization After the logical data model is optimized in terms of data storage, the project team may decide to denormalize, or add redundancy back into the design that is depicted in the physical data model. *Denormalization* reduces the number of joins that must be performed in a query, thus speeding up data access. Figure 11-17 shows a denormalized physical data model for customer orders.

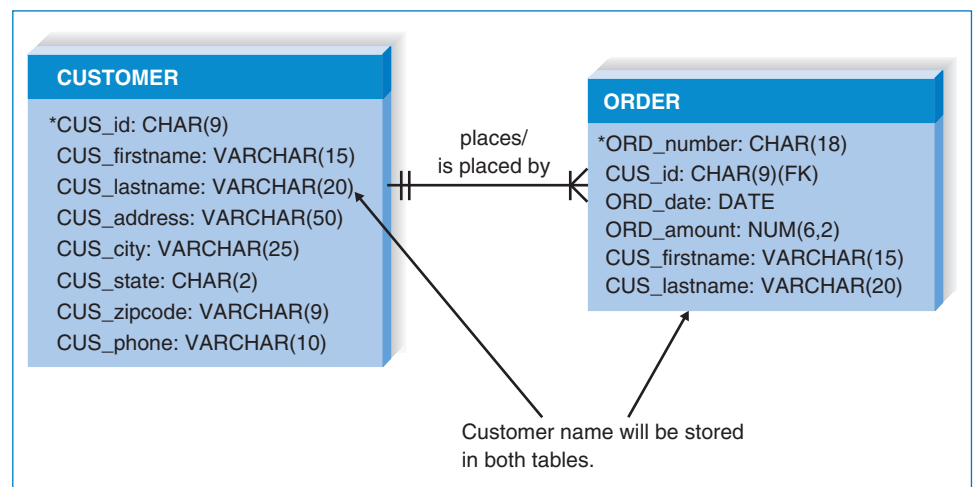


FIGURE 11-17
Denormalized Data Model

YOUR

TURN

11-4 DENORMALIZING A STUDENT ACTIVITY FILE

Consider the logical data model that you created in Chapter 6 for “Your Turn 6-7.” Examine the model and describe possible opportunities for denormalization.

QUESTION:

How would you denormalize the physical data model, and what are the benefits of your changes?

The customer name was added back into the Order table because the project team learned during the analysis phase that queries about orders usually require the customer’s name. Instead of joining the Order table repeatedly to the Customer table, the system now needs to access only the Order table because it contains all the relevant information.

Of course, denormalization should be applied sparingly, for the reasons described in the previous section, but it is ideal in situations in which information is queried frequently yet updated rarely. There are four cases in which you may rely on denormalization to reduce joins and improve performance (see Figure 11-18). First, denormalization can be applied in the case of *look-up tables*, which are tables that contain descriptions of values (e.g., a table of product descriptions, a table of payment types). Because descriptions of codes rarely change, it may be more efficient to include the description along with its respective code in the main table, to eliminate the need to join the look-up table each time a query is performed.

Second, 1:1 relationships are good candidates for denormalization. Although logically, two entities should be separated, from a practical standpoint the information from both entities may be regularly accessed together. Think about an order and its shipping information. Logically, it may make sense to separate the attributes related to shipping into a separate entity, but as a result, the queries regarding shipping likely will always need a join to the Order table. If the project team finds that certain shipping information, such as state and shipping method, are needed when orders are accessed, they may decide to combine the entities or include some shipping attributes in the order entity.

Third, at times it will be more efficient to include a parent entity’s attributes in its child entity on the physical data model. For example, consider a customer table and an order table that share a 1:N relationship, with customer as the parent and order as the child. If queries regarding orders continuously require customer information, the most popular customer fields can be placed in order to reduce the required joins to the customer table, as was done with customer name in Figure 11-18.

Finally, denormalization is applied when a popular data modeling technique called *star schema design* is used.² Learning how to model with star schema is beyond the scope of this book, but there are a number of Web resources and books available that are listed on the textbook Web site. Basically, star schema is a way to model data whereby the data are denormalized to speed up data access for DSS. It uses two kinds of tables—fact tables and dimension tables—to store numerical, additive, and descriptive data, respectively. Star schema modeling is the way in

² A good book on star schema design is that by Claudia Imhoff, Nicholas Gallemmo, and Jonathan Geiger, *Mastering Data Warehouse Design: Relational and Dimensional Techniques*, John Wiley & Sons, 2003.

| Reason | Description | Example |
|--------------------|--|---------|
| Look-up Table | Include a code's description in the table using that code if the description is often used. | |
| 1:1 Relationships | Combine tables if they are related 1:1 and if they usually are accessed together. | |
| 1:N Relationships | Place fields from the parent (1) table into the child (N) table if the parent fields are used frequently with child information. | |
| Star Schema Design | Data marts often are modeled with star schema design, which uses denormalization to maximize DSS query performance. | |

FIGURE 11-18
Reasons to Denormalize

which relational databases can be designed to emulate a multidimensional database. See Figure 11-18 for an example of a star schema design of a customer order database. The fact table contains order amount and cost (i.e., additive data), and the other tables contain information describing different dimensions of an order: the customer, the order itself, and time.

Clustering Speed of access also is influenced by the way in which the data are retrieved. Think about going shopping in a grocery store. If you have a list of items to buy, but you are unfamiliar with the store's layout, then you need to walk down every aisle to make sure that you don't miss anything from your list. Likewise, if records are arranged on a hard disk in no particular order (or in an order that is unrelated to your data needs), then any query of the records results in a *table scan* in which the DBMS has to access every row in the table before retrieving the result set. Table scans are the most inefficient of data retrieval methods.

One way to improve access speed is to reduce the number of times that the storage medium must be accessed during a transaction. This can be accomplished by *clustering* records together physically so that like records are stored close together. With *intrafile clustering*, similar records in the table are stored together in some way, such as in order by primary key or, in the case of a grocery store, by item type. Thus, whenever a query looks for records, it can go directly to the right spot on the hard disk (or other storage medium) because it knows in what order the records are stored, just as we can walk directly to the bread aisle in the store to pick up a loaf of bread. *Interfile clustering* combines records from more than one table that typically are retrieved together. For example, if customer information is usually accessed with the related order information, then the records from the two tables may be physically stored in a way that preserves the customer order relationship. Returning to the grocery store scenario, an interfile cluster would be similar to storing peanut butter, jelly, and bread next to each other in the same aisle because they are usually purchased together, not because they are similar types of items. Of course, each table can have only one clustering strategy because the records can be arranged physically in only one way.

CONCEPTS

11-A MAIL-ORDER INDEX

IN ACTION

A Virginia-based mail-order company sends out approximately 25 million catalogs each year, using a customer table with 10 million names. Although the primary key of the customer table is customer identification number, the table also contains an index of customer last names. Most people who call to place orders remember their last name, but not their customer identification number, so this index is used frequently.

An employee of the company explained that indexes are critical to reasonable response times. A fairly complicated query was written to locate customers by the

state in which they lived, and it took over three weeks to return an answer. A customer state index was created, and that same query provided a response in 20 minutes; that's 1512 times faster!

QUESTION:

As an analyst, how can you make sure that the proper indexes have been put in place so that users are not waiting for weeks to receive the answers to their questions?

Indexing A time saver that you are familiar with is an index located in the back of a textbook, which points you directly to the page or pages that contain your topic of interest. Think of how long it would take to find all of the times that *relational database* appears in this textbook if you didn't have the index to rely on. An *index* in data storage is like an index in the back of a textbook; it is a minitable that contains values from one or more columns in a table and the location of the values within the table. Instead of paging through the entire textbook, you can move directly to the right pages and get the information you need. Indexes are one of the most important ways to improve database performance. Whenever you have performance problems, the first place to look is an index.

A query can use an index to find the locations of only those records that are included in the query answer, and a table can have an unlimited number of indexes. Figure 11-19 shows an index that orders records by payment type. A query that searches for all of the customers who used American Express can use this index to find the locations of the records that contain American Express as the payment type without having to scan the entire order table.

Project teams can make indexes perform even faster by placing them into the main memory of the data storage hardware. Retrieving information from memory is much faster than from another storage medium like a hard disk—think about how much faster it is to retrieve a phone number that you have memorized versus one that you need to look up in a phone book. Similarly, when a database has an index in memory, it can locate records very, very quickly.

Of course, indexes require overhead in that they take up space on the storage medium. Also, they need to be updated as records in tables are inserted, deleted, or changed. Thus, although indexes lead to faster access to the data, they slow down the update process. In general, you should create indexes sparingly for transaction systems or systems that require a lot of updates, but apply indexes generously when designing systems for decision support (Figure 11-20).

Usually, CASE tools allow you to define indexes and clustering strategies within the design of the physical data model. Figure 11-21 shows the index screen in one CASE tool (ERwin) for the order table. In this example, three

| Payment Type | Pointer |
|--------------|---------|
| AMEX | * |
| AMEX | * |
| AMEX | * |
| AMEX | * |
| AMEX | * |
| MC | * |
| MC | * |
| MC | * |
| MC | * |
| MC | * |
| MC | * |
| VISA | * |
| VISA | * |
| VISA | * |
| VISA | * |
| VISA | * |
| VISA | * |
| VISA | * |

| Order Number | Date | Cust ID | Amount | Payment Type |
|--------------|----------|---------|---------|--------------|
| 234 | 11/23/11 | 4254 | \$30.00 | MC |
| 235 | 11/23/11 | 9500 | \$20.00 | VISA |
| 236 | 11/23/11 | 1556 | \$20.00 | VISA |
| 237 | 11/23/11 | 2487 | \$60.00 | AMEX |
| 238 | 11/23/11 | 2243 | \$50.00 | MC |
| 239 | 11/23/11 | 1035 | \$50.00 | AMEX |
| 240 | 11/23/11 | 1556 | \$20.00 | VISA |
| 241 | 11/23/11 | 1123 | \$40.00 | MC |
| 242 | 11/24/11 | 9501 | \$30.00 | VISA |
| 243 | 11/24/11 | 4453 | \$30.00 | VISA |
| 244 | 11/24/11 | 9505 | \$20.00 | VISA |
| 245 | 11/24/11 | 2282 | \$20.00 | AMEX |
| 246 | 11/24/11 | 5927 | \$60.00 | MC |
| 247 | 11/24/11 | 2241 | \$50.00 | VISA |
| 248 | 11/24/11 | 4254 | \$50.00 | AMEX |
| 249 | 11/24/11 | 2242 | \$50.00 | AMEX |
| 250 | 11/24/11 | 2274 | \$20.00 | VISA |
| 251 | 11/24/11 | 9507 | \$10.00 | VISA |
| 252 | 11/24/11 | 2487 | \$60.00 | VISA |
| 253 | 11/24/11 | 2264 | \$40.00 | AMEX |

FIGURE 11-19
Payment Type Index

FIGURE 11-20
Guidelines for Creating Indexes

- Use indexes sparingly for transaction systems.
- Use many indexes to improve response times in decision support systems.
- For each table, create a unique index that is based on the primary key.
- For each table, create an index that is based on the foreign key to improve the performance of joins.
- Create an index for fields that are used frequently for grouping, sorting, or criteria.

indexes have been designed for the table, and during the implementation phase, the CASE tool will generate the code that is necessary to construct these indexes in the DBMS.

Estimating Storage Size

Even if you have denormalized your physical data model, clustered records, and created indexes appropriately, the system will perform poorly if the database server cannot handle its volume of data. Therefore, one last way to plan for good performance is to apply *volumetrics*, which means estimating the amount of data that

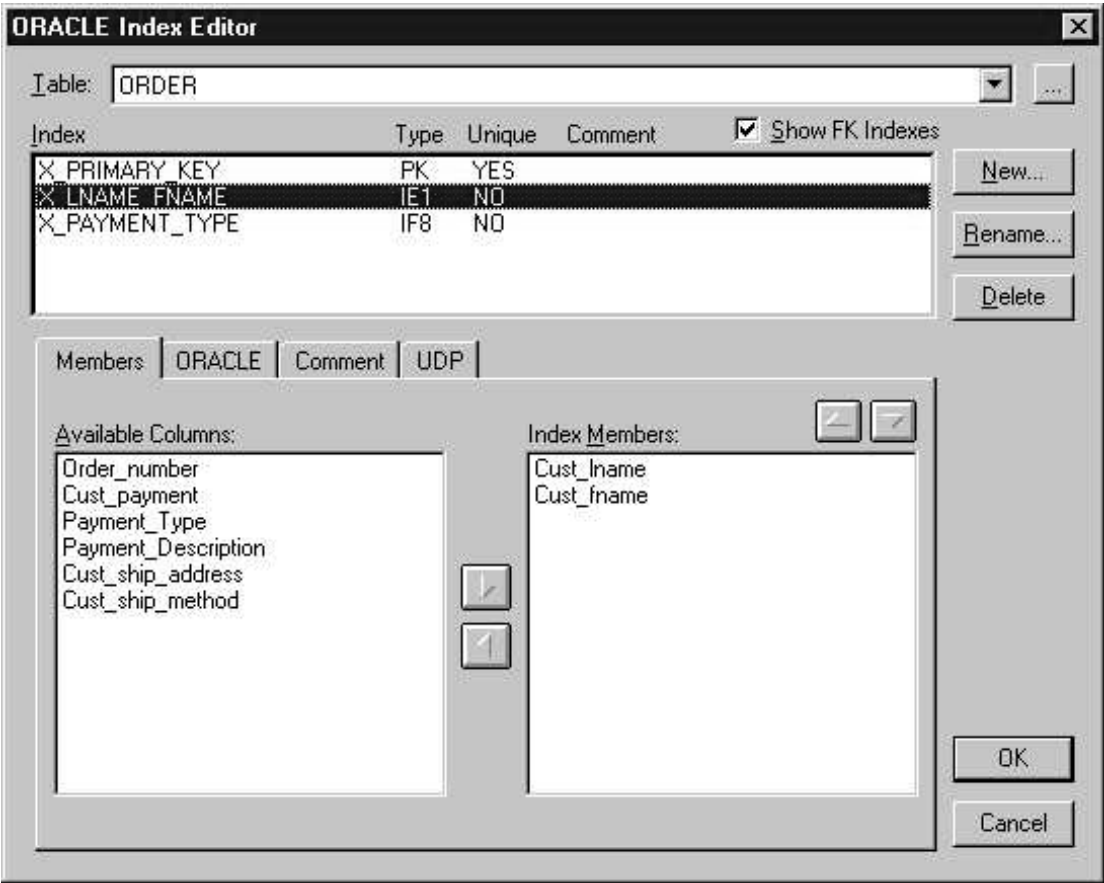


FIGURE 11-21
Indexes in ERwin

the hardware will need to support. You can incorporate your estimates into the database server hardware specification to make sure that the database hardware is sufficient for the project's needs. The size of the database will be determined by the amount of raw data in the tables and the *overhead* requirements of the DBMS. To estimate size, you will need to have a good understanding of the initial size of your data as well as its expected growth rate over time.

Raw data refers to all the data that are stored within the tables of the database, and it is calculated via a bottom-up approach. First, write down the estimated average width for each column (field) in the table and sum the values, yielding a total record size (Figure 11-22). For example, if a variable-width last name column is assigned a width of 20 characters, you can enter 13 as the average character width of the column. In Figure 11-22, the estimated record size is 49.

Next, calculate the overhead for the table as a percentage of each record. Overhead includes the room needed by the DBMS to support such functions as administrative actions and indexes, and it should be assigned on the basis of past experience, recommendations from technology vendors, or parameters that are built into software written to calculate volumetrics. For example, your DBMS vendor may recommend that you allocate 30% of the records' raw data size for overhead storage space, creating a total record size of 63.7 characters in the Figure 11-22 example.

Finally, record the number of initial records that will be loaded into the table, as well as the expected growth per month. This information should have been collected during the analysis phase as a nonfunctional data requirement. As shown in Figure 11-22, the initial space required by the first table is 3,185,000 characters, and future sizes can be projected on the basis of the growth figure. These steps are repeated for every table in order to get a total size for the entire database.

Many CASE tools will provide you with database size information on the basis of how you set up the physical data model, and they will calculate volumetrics estimates automatically. Figure 11-23 shows a volumetrics screen for ERwin.

Ultimately, the size of the database needs to be shared with the design team so that the proper technology can be put in place to support the system's data, and

| Field | Average Size (Characters) |
|--------------------------|---------------------------|
| Order number | 8 |
| Date | 7 |
| Cust ID | 4 |
| Last name | 13 |
| First name | 9 |
| State | 2 |
| Amount | 4 |
| Tax rate | 2 |
| Record size | 49 |
| Overhead | 30% |
| Total record size | 63.7 |
| Initial table size | 50,000 |
| Initial table volume | 3,185,000 |
| Growth rate/month | 1000 |
| Table volume @ 3 years | 5,478,200 |

FIGURE 11-22
Calculating Volumetrics

Volumetrics Editor : Order and Shipment Tables

Settings | Report | Parameters

Table:
ORDER
SHIPMENT_INFO

Sizing Input:

Table Row Counts
Initial: 250000 Max: 0 Grow By: 2500 per Month

Column Properties:

| Column | Datatype | Alloc | Avg Width | Pct NULL |
|-------------------|--------------|-------|-----------|----------|
| Cust_fname | CHAR(15) | 15 | | |
| Cust_lname | CHAR(20) | 20 | | |
| Cust_payment | DECIMAL(5,2) | 0 | 0 | 0 |
| Cust_ship_address | CHAR(40) | 40 | | 0 |
| Cust_ship_method | CHAR(4) | 4 | | 25 |
| Order_number | CHAR(8) | 8 | | |

Sizing Estimates
Average Row Size 111
Initial Table Size: 26 M
Initial size of Index(es): 12 M

Include Indexes
☒ PK ☒ AK
☒ FK ☒ IE

Storage
Physical Object: ...
<default>

Close

(a)

Volumetrics Editor : Order and Shipment Tables

Settings | Report | Parameters

Options

Report
☐ Physical Objects
☐ Database Objects
☒ Tables

Time
☐ Initial
☒ Projections
Months: 12

Send to Report Browser...

Physical Objects:

| Table/Index | Size | Physical Object |
|--------------------|-------------|-----------------|
| ORDER | 30 M | <default> |
| X_PRIMARY_KEY | 2324 K | <default> |
| X_LNAME_FNAME | 9 M | <default> |
| X_PAYMENT_TYPE | 1230 K | <default> |
| Total | 42 M | |
| SHIPMENT_INFO | 16 M | <default> |
| XPKSHIPMENT_INFO | 2324 K | <default> |
| XIFSHIPMENT_INFO | 2324 K | <default> |
| Total | 21 M | |
| Grand Total | 63 M | |

Close

(b)

FIGURE 11-23

Volumetrics Screen in ERwin: (a) Information about Columns and Rows Is Entered into the ERwin; (b) Report Is Generated on the Basis of the Information.

potential performance problems can be addressed long before they affect the success of the system.

Applying the Concepts at Tune Source

Now that the team members had a good idea of the type of data storage formats that would be used, they were ready to optimize the data for performance efficiency. Kenji was the analyst in charge of the logical data model, and Jason wanted to be sure that the data model was optimized for storage efficiency before the team discussed access speed issues.

Kenji assured Jason that the current data model was in third normal form. He was confident of this because the project team followed the data modeling guidelines that led to a well-formed logical model. Of course, a week or so earlier, he did apply the three rules of normalization to the data model as a check to make sure that no design errors were overlooked.

Kenji then asked about the file formats for the transaction file identified in the earlier meeting. Jason suggested that he normalize the files to better understand the various tables that would be involved in the import procedure.

The last step of data storage design was to optimize the design for data access speed. Jason met with the analysts on the data storage design team and talked about the techniques that were available to speed up access to data in the system. Together, the team listed all the data that would be supported by the Digital Music Download system and discussed how all the data would be used. They developed the strategy laid out in Figure 11-24 to identify the specific techniques to put in place.

Ultimately, clustering strategies, indexes, and denormalization decisions were applied to the physical data model, and a volumetrics report was run from the CASE tool to estimate the initial and projected sizes of the database. The report suggested that an initial data storage capacity of about 5 gigabytes would be needed for the expected one-year life of the first version of the system. Additional storage capacity would be needed as the number of available tunes increases and for future versions of the system.

| Target | Comments | Suggestions to Improve Data Access Speed |
|-----------------------|---|--|
| All tables | Basic table manipulation | Investigate whether records should be clustered physically by primary key. Create indexes for primary keys. Create indexes for foreign key fields. |
| All tables | Sorts and grouping | Create indexes for fields that are frequently sorted or grouped. |
| Tune information | Users will need to search Tune information by title, artist, and genre. | Create indexes for Tune title, artist, and genre. |
| Entire physical model | Investigate denormalization opportunities for all fields that are not updated very often. | Investigate 1:1 relationships. Investigate look-up tables. Investigate 1:M relationships. |

FIGURE 11-24
Digital Music Download System
Performance

Jason gave the estimates to the analyst in charge of managing the server hardware acquisition so that the person could ensure that the technology could handle the expected volume of data for the Digital Music Download system. The estimates would also go to the DBMS vendor during the implementation of the software so that the DBMS could be configured properly.

SUMMARY

File Data Storage Formats

There are two basic types of data storage formats: files and databases. Files are electronic lists of data that have been optimized to perform a particular transaction, and there are five different types: master, look-up, transaction, audit, and history. Master files typically are kept for long periods because they store important business information, such as order information or customer mailing information. Look-up files contain static values that are used to validate fields in the master files, and transaction files temporarily hold information that will be used for a master file update. An audit file records “before” and “after” images of data as they are altered so that an audit can be performed if the integrity of the data is questioned. Finally, the history file stores past transactions (e.g., old customers, past orders) that are no longer needed by the system.

Database Data Storage Formats

A database is a collection of groupings of information that are related to each other in some way, and a DBMS (database management system) is software that creates and manipulates these databases. There are four types of databases that are likely to be encountered during a project: legacy, relational, object, and multidimensional. The legacy databases (e.g., hierarchical databases and network databases) use older, sometimes outdated technology and are rarely used to develop new applications. The relational database is the most popular kind of database for application development today, and it is based on collections of tables that are related to each other through common fields known as foreign keys. Object databases contain data and processes that are represented by object classes, and relationships between object classes are shown by encapsulating one object class within another and are mainly used in multimedia applications (e.g., graphics, video, and sound). One of the newest members in the database arena is the multidimensional database, which has become popular with the increase in data warehousing. It stores precalculated quantitative information (e.g., totals, averages) at the intersection of dimensions (e.g., time, salesperson, product) to support applications that require data to be sliced and diced.

Selecting a Data Storage Format

The application’s data should drive the storage format decision. Relational databases support simple data types very effectively, whereas object databases are best for complex data. Multidimensional databases are tuned to store aggregated, quantitative information. The type of system also should be considered when choosing among data storage formats. Relational databases have matured to support transactional systems, whereas multidimensional databases have been designed to perform best in decision support environments. Although less critical to the format selection decision, the project team needs to consider the kind of technology that exists within the organization and the kind of technology likely to be used in the future.

Physical Entity Relationship Diagrams

One important aspect of design is the movement from logical to physical entity relationship diagrams. Physical ERDs contain references to how data will be stored in a file or database table, and metadata are included to describe the data model components. The model reflects design decisions that will affect the physical implementation of the system.

The CRUD matrix should be modified to show exactly how data in the physical data model are created and used in the physical process model. The CRUD matrix helps ensure balance between the physical process and data models prior to implementation.

Optimizing Data Storage

There are two primary dimensions in which to optimize a relational database: for storage efficiency and for speed of access. The most efficient relational database tables in terms of data storage are those that have no redundant data and very few null values. Normalization is the process whereby a series of rules are applied to the logical data model to determine how well optimized it is for storage efficiency.

Once you have optimized your logical data design for storage efficiency, the data may be spread out across a number of tables. To improve speed, the project team may decide to denormalize—or add redundancy back into—the design that is depicted in the physical data model. Denormalization reduces the number of joins that must be performed in a query, thus speeding up data access. Denormalization is best in situations in which data are accessed frequently and updated rarely. There are three modeling situations that are good candidates for denormalization: look-up tables, entities that share one-to-one (1:1) relationships, and entities that share one-to-many (1:M) relationships. In all three cases, attributes from one entity are moved or repeated in another entity to reduce the joins that must occur during data access.

Clustering occurs when similar records are stored close together on the storage medium to speed up data retrieval. In intrafile clustering, similar records in the table are stored together in some way, such as in sequence. Interfile clustering combines records from more than one table that typically are retrieved together. Indexes also can be created to improve the access speed of a system. An index is a minitable that contains values from one or more columns in a table and information about where the values can be found. Instead of performing a table scan, which is the most inefficient way to retrieve data from a table, an index points directly to the records that match the requirements of a query.

Finally, the speed of the system can be improved if the right hardware is purchased to support its data. Analysts can use volumetrics to estimate the current and future size of data in the database and then share these numbers with the people who are responsible for buying and configuring the database hardware.

KEY TERMS

| | | |
|------------------|-----------------------------------|------------------------------------|
| Aggregated | Database management system (DBMS) | End-user DBMS |
| Audit file | Decision support system (DSS) | Enterprise DBMS |
| Clustering | Default value | Executive information system (EIS) |
| Data mart | Denormalization | Expert system (ES) |
| Data warehousing | Encapsulation | File |
| Database | | Foreign key |

| | | |
|-------------------------------------|--------------------------------------|---------------------------------|
| Hierarchical database | Master file | Raw data |
| History file | Member | Referential integrity |
| Hybrid object-oriented DBMS | Multidimensional database | Relational database |
| Index | Network database | Set |
| Instantiation | Normalization | Star schema design |
| Interfile cluster | Object class | Structured Query Language (SQL) |
| Intrafile cluster | Object database | Subclass |
| Legacy database | Object-oriented DBMS | Table scan |
| Linked list | Overhead | Transaction file |
| Logical entity relationship diagram | Physical data model | Transaction processing system |
| Look-up file | Physical entity relationship diagram | Valid value |
| Look-up table | Pointer | Volumetrics |
| Management information system (MIS) | Primary key | |

QUESTIONS

- Describe the two steps to data storage design.
- How are a file and a database different from each other?
- What is the difference between an end-user database and an enterprise database? Provide an example of each one.
- Name five types of files, and describe the primary purpose of each type.
- Name two types of legacy databases and the main problems associated with each type.
- What is the most popular kind of database today? Provide three examples of products that are based on this technology.
- What is referential integrity, and how is it implemented in a relational database?
- What is the biggest strength of the object database? Describe two of its weaknesses.
- How does the multidimensional database store data?
- What are the two most important factors in determining the type of data storage format that should be adopted for a system? Why are these factors so important?
- Why should you consider the storage formats that already exist in an organization when deciding on a storage format for a new system?
- What are the differences between the logical and physical ERDS?
- Describe the metadata associated with the physical ERD.
- Describe the purpose of the primary and foreign keys.
- Name three ways that null values in a database can be interpreted. Why is this problematic?
- What are the two dimensions in which to optimize a relational database?
- What is the purpose of normalization?
- Describe three situations that can be good candidates for denormalization.
- Describe several techniques that can improve performance of a database.
- What is the difference between interfile and intrafile clustering? Why are they used?
- What is an index, and how can it improve the performance of a system?
- Describe what should be considered when estimating the size of a database.
- Why is it important to understand the initial and projected size of a database during the design phase?
- What are the key issues in deciding between using perfectly normalized databases and denormalized databases?

EXERCISES

- Using the Web or other resources, identify a product that can be classified as an end-user database and a product that can be classified as an enterprise database. How are the products described and marketed? What kinds of applications and users do they support? In what kinds of situations would an organization choose to implement an end-user database over an enterprise database?

- B. Visit a commercial Web site (e.g., CDnow, Amazon.com). If files were being used to store the data supporting the application, what types of files would be needed? What data would they contain?
- C. Using the Web, review one of the products listed at the end of this exercise. What are the main features and functions of the software? In what companies has the database management system (DBMS) been implemented, and for what purposes? According to the information that you found, what are three strengths and weaknesses of the product?

- Relational DBMS
- Object DBMS
- Multidimensional DBMS

- D. You have been given a file that contains fields relating to CD information. Using the steps of normalization, create a logical data model that represents this file in third normal form. The fields include the following:

- Musical group name
- Musicians in group
- Date group was formed
- Group's agent
- CD title 1
- CD title 2
- CD title 3
- CD 1 length
- CD 2 length
- CD 3 length

The assumptions are as follows:

- Musicians in group contains a list of the members in the musical group.
 - Musical groups can have more than one CD, so both group name and CD title are needed to uniquely identify a particular CD.
- E. Jim Smith's dealership sells Fords, Hondas, and Toyotas. The dealership keeps information about each car manufacturer with whom it deals so that employees can get in touch with manufacturers easily. The dealership staff also keeps information about the models of cars that the dealership carries from each manufacturer. They keep such information as list price, the price the dealership paid to obtain the model, and the model name and series (e.g., Honda Civic LX). They also keep information about all sales that they have made. (For instance, they will record the buyer's name, the car he or she bought,

and the amount he or she paid for the car.) So that staff can contact the buyers in the future, contact information is also kept (e.g., address, phone number). Create a logical data model. (You may have done this already in Chapter 7.) Apply the rules of normalization to the model to check the model for processing efficiency.

- F. Describe how you would denormalize the model that you created in question E. Draw the new physical model on the basis of your suggested changes. How would performance be affected by your suggestions?
- G. Examine the physical data model that you created in question F. Develop a clustering and indexing strategy for this model. Describe how your strategy will improve the performance of the database.
- H. Investigate the volumetric interface with the computer-aided software engineering (CASE) tool that you are using for class. What information do you as an analyst need to input into the tool? How are size estimates calculated? If your CASE tool does not accept volumetric information, how can you calculate the size of the database?
- I. Calculate the size of the database that you created in question F. Provide size estimates for the initial size of the database as well as for the database in one year's time. Assume that the dealership sells 10 models of cars from each manufacturer to approximately 20,000 customers a year. The system will be set up initially with one year's worth of data.
- J. How would the following ERD be changed to incorporate the design decision listed next?

- The analyst wants to keep track of the user ID of anyone who changes a grade for a course.
- A data store is added on the physical DFD so that information regarding the current semester's courses can be stored temporarily during the add/drop period before the courses become a part of the student's permanent record.
- The system would like to archive alumni into a table, once they graduate, so that only active students are stored in the student table.

- K. Draw a physical process model (just the processes and data stores) for the following CRUD matrix:

| | Register | Schedule | Create | Create |
|--------------------|----------|----------|------------|--------|
| Student | Student | Student | Transcript | Bill |
| Student Data Store | CRUD | R | R | R |
| Course Data Store | | CRUD | R | |
| Billing Data Store | | CRUD | | CRUD |
| Grade Data Store | | | CRUD | |

MINICASES

1. In the new system under development for Holiday Travel Vehicles, seven tables will be implemented in the new relational database. These tables are New Vehicle, Trade-in Vehicle, Sales Invoice, Customer, Salesperson, Installed Option, and Option. The expected average record size for these tables and the initial record count per table are given next.

| Table Name | Average Record Size | Initial Table Size (records) |
|------------------|------------------------|---------------------------------|
| New Vehicle | 65 characters | 10,000 |
| Trade-in Vehicle | 48 characters | 7,500 |
| Sales Invoice | 76 characters | 16,000 |
| Customer | 61 characters | 13,000 |
| Salesperson | 34 characters | 100 |
| Installed Option | 16 characters | 25,000 |
| Option | 28 characters | 500 |

Perform a volumetrics analysis for the Holiday Travel Vehicles system. Assume that the DBMS that will be used to implement the system requires 35% overhead to be factored into the estimates. Also, assume a growth rate for the company of 10% per year. The systems development team wants to ensure that adequate hardware is obtained for the next three years.