OCW: DATABASE MANAGEMENT SYSTEMS (EC703C)

DATABASE MANAGEMENT SYSTEM
EC 703C
Contact: 3L
Credits: 3

Prepared by: Palasri Dhar, Antara Ghosal

Prerequisite:

1. Logic of programming language
2. Basic concepts of data structure and algorithms

Course Objectives

1. To learn the data models, conceptualize and depict a database system
2. To design system using E-R diagram.
3. To learn SQL & relational database design.
4. To understand the internal storage structures using different file and indexing techniques.
5. To know the concepts of transaction processing, concurrency control techniques and recovery procedure**.**

Module 1:
Introduction [3L]

Concept & Overview of DBMS, Data Models, Database Languages, Database Administrator, Database Users, Three Schema architecture of DBMS.

Module 2:
Entity-Relationship and Relational Database Model [11L]
Basic concepts, Design Issues, Mapping Constraints, Keys, Entity-Relationship Diagram, Weak Entity Sets, Extended E-R features, case study on E-R Model. Structure of relational Databases, Relational Algebra, Relational Calculus, Extended Relational Algebra Operations, Views, Modifications Of the Database.

Module 3:
SQL and Integrity Constraints [6L]
Concept of DDL, DML, DCL. Basic Structure, Set operations, Aggregate Functions, Null Values, Domain Constraints, Referential Integrity Constraints, assertions, views, Nested Subqueries, Database security application development using SQL, Stored procedures and triggers.
Module 4:
Relational Database Design [8L]
Functional Dependency, Different anomalies in designing a Database., Normalization using functional dependencies, Decomposition, Boyce-Codd Normal Form, 3NF, Normalization using multi-valued dependencies, 4NF, 5NF , Case Study

Module 5:

Internals of RDBMS [9L]

Physical data structures, Query optimization: join algorithm, statistics and cost bas optimization. Transaction processing, Concurrency control and Recovery Management: transaction model properties, state serializability, lock base protocols; two phase locking, Dead Lock handling

Module 6:

File Organization & Index Structures [6L]

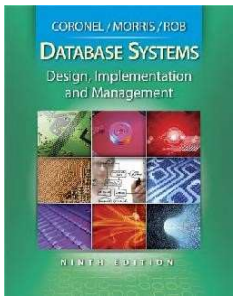
File & Record Concept, Placing file records on Disk, Fixed and Variable sized Records, Types of Single-Level Index (primary, secondary, clustering), Multilevel Indexes **Text Books:**

1. Henry F. Korth and Silberschatz Abraham, "Database System Concepts", Mc.Graw Hill.
2. Elmasri Ramez and Novathe Shamkant, "Fundamentals of Database Systems", Benjamin Cummings

Publishing. Company.

3. Ramakrishnan: Database Management System , McGraw-Hill
4. Gray Jim and Reuter Address, "Transaction Processing : Concepts and Techniques", Moragan Kauffman Publishers. 5. Ullman JD., "Principles of Database Systems", Galgottia Publication.

Reference:



1. Jain: Advanced Database Management System CyberTech
2. Date C. J., "Introduction to Database Management", Vol. I, II, III, Addison Wesley.
3. "Fundamentals of Database Systems", Ramez Elmasri, Shamkant B.Navathe, Addison Wesley Publishing Edition
4. "Database Management Systems", Arun K.Majumdar, Pritimay Bhattacharya, Tata McGraw Hill

Course Outcomes (COs)

On completion of the course students will be able to

1. Apply the knowledge of Entity Relationship (E-R) diagram for an application.
2. Create a normalized relational database model
3. Analyze real world queries to generate reports from it.
4. Determine whether the transaction satisfies the ACID properties.
5. Create and maintain the database of an organization.

CO-PO MAPPING

CO #	PO1	PO2	PO3	PO4	PO5	PO6	PO7	PO8	PO9	PO10	PO11	PO12
CS(EE)705D.1	2	2	2	2	3	2	1	1	2	2	3	3
CS(EE)705D.2	2	3	3	3	3	1	1	1	2	2	3	3
CS(EE)705D.3	3	3	2	3	3	2	2	2	3	3	3	3
CS(EE)705D.4	3	3	2	2	2	1	1	1	1	1	2	3
CS(EE)705D.5	3	3	3	3	3	2	2	2	3	3	3	3
CS(EE)705D(average)	3	3	2	3	3	2	1	1	2	2	3	3

3=HIGH, 2= MEDIUM, 1=LOW

DATABASE MANAGEMENT SYSTEM

DESIGN, IMPLEMENTATION AND MANAGEMENT

Composed By Aniruddha Biswas, Asst. Professor, IT, JISCE

BRIEF CONTENTS

Module 2: Entity-Relationship and Relational Database Model

Chapter 3: The Relational Database Model
Chapter 4: Entity Relationship (ER) Modeling
Chapter 5: Advanced Data Modeling

Module 3: SQL and Integrity Constraints

Chapter 6: Introduction to Structured Query Language (SQL)
Chapter 7: Advanced SQL
Chapter 8: Database Design

Module 4: Relational Database Design

Chapter 9: Normalization of Database Tables

Module 5: Internals of RDBMS

Chapter 10: Transaction Management and Concurrency Control
Chapter 11: Database Performance Tuning and Query Optimization

Module 6: File Organization & Index Structures

Chapter 12: Database File and Indexing

TABLE OF CONTENTS

Module 1: Introduction

Chapter 1: Database Systems

Chapter 2 Data Models

Module 2: Entity-Relationship and Relational Database Model

Chapter 3: The Relational Database Model

Chapter 4: Entity Relationship (ER) Modeling

Chapter 5: Advanced Data Modeling

Module 3: SQL and Integrity Constraints

Chapter 6: Introduction to Structured Query Language (SQL)

Chapter 7: Advanced SQL

Chapter 8: Database Design

Module 4: Relational Database Design

Chapter 9: Normalization of Database Tables

Module 5: Internals of RDBMS

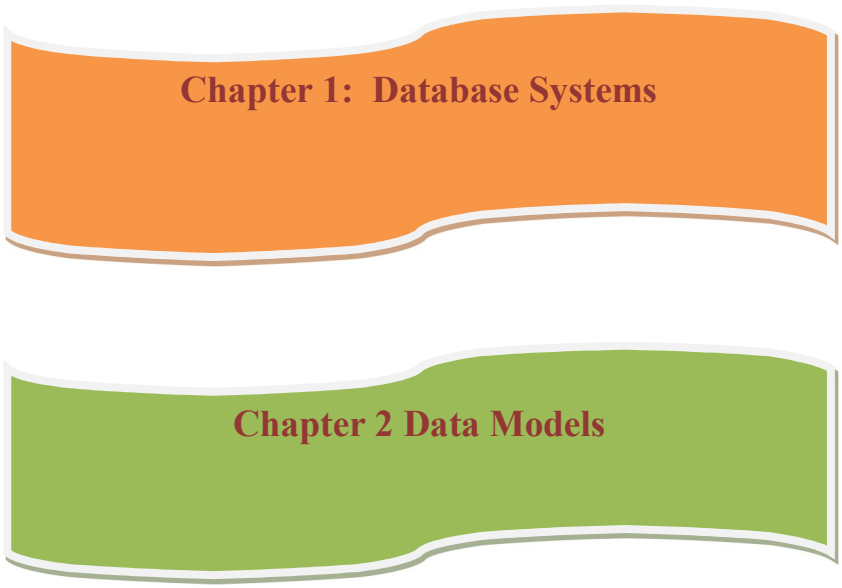
Chapter 10: Transaction Management and Concurrency Control

Chapter 11: Database Performance Tuning and Query Optimization

Module 6: File Organization & Index Structures

Chapter 12: Database File and Indexing

Module 1: Introduction



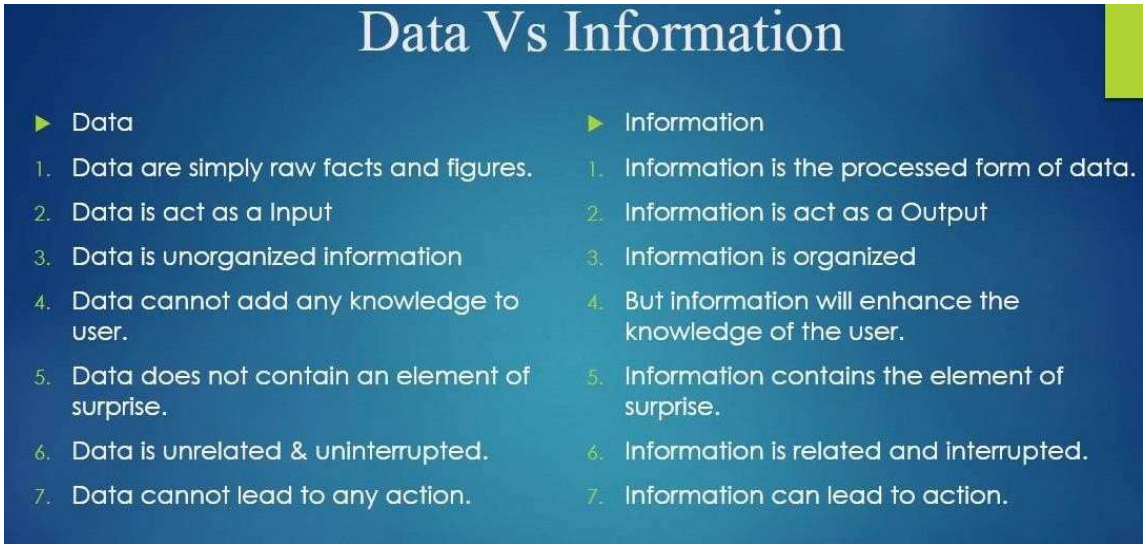
Chapter 1: Database Systems

1.1 WHY DATABASES?

In our daily life we deal with various kind of data and internally in our brain we organize them and work upon them in time sometimes fail. But for a large organization it becomes a really hectic job to organize big chunks of data in the brain of the decision makers. At the core of all these organizational systems are *collection, extraction, transformation, storage, aggregation, manipulation, dissemination, managing and reporting*. If we talk about the size of these data which might be beyond our imagination may vary from terra bytes to yotta bytes. *How can these businesses process this much data? How can they store it all, and then quickly retrieve just the facts that decision makers needs, just when they want to have it? The answer is that they use databases.*

1.2 DATA VS. INFORMATION

To understand what drives database design, you must understand the difference between data and information. *Data are raw facts*. The word raw indicates that the facts have not yet been processed to reveal their meaning. Organized, processed and meaningful set of data is called Information.



1.3 INTRODUCING THE DATABASE

Efficient data management typically requires the use of a computer database. A database is a shared, integrated computer structure that stores a collection of:

_ End-user data, that is, raw facts of interest to the end user.

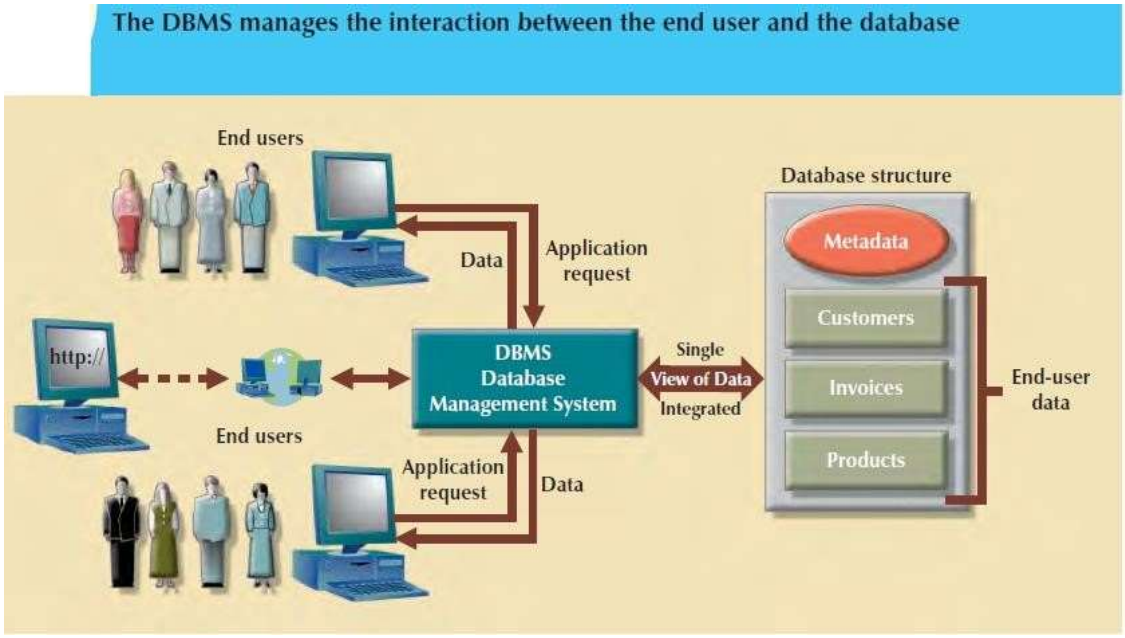
_ Metadata, or data about data, through which the end-user data are integrated and managed.

The metadata provide information that complements and expands the value and use of the data. In short, metadata present a more complete picture of the data in the database. Given the characteristics of metadata, you might hear a database described as a "collection of self-describing data." *A database management system (DBMS) is a collection of programs that manages the database structure and controls access to the data stored in the database.*

1.3.1 ADVANTAGES OF THE DBMS

The DBMS serves as the intermediary between the user and the database. The database structure itself is stored as a collection of files, and the only way to access the data in those files is through the DBMS. Below figure emphasizes the point that the DBMS presents the end user (or application program) with a single, integrated view of the data in the database. The DBMS receives all application requests and translates them into the complex operations required to fulfill those requests. The DBMS hides much of the database's internal complexity from the application programs and users. The application

program might be written by a programmer using a programming language such as Visual Basic.NET, Java, or C#, or it might be created through a DBMS utility program.



The DBMS helps make data management more efficient and effective. In particular, a **DBMS provides advantages such as:**

- **Improved data sharing.** The DBMS helps create an environment in which end users have better access to more and better-managed data. Such access makes it possible for end users to respond quickly to changes in their environment.

- **Improved data security.** The more users access the data, the greater the risks of data security breaches. Corporations invest considerable amounts of time, effort, and money to ensure that corporate data are used properly. A DBMS provides a framework for better enforcement of data privacy and security policies.

- **Better data integration.** Wider access to well-managed data promotes an integrated view of the organization's operations and a clearer view of the big picture. It becomes much easier to see how actions in one segment of the company affect other segments.

- **Minimized data inconsistency.** Data inconsistency exists when different versions of the same data appear in different places. For example, data inconsistency exists when a company's sales department stores a sales representative's name as "Bill Brown" and the company's personnel department stores that same person's name as "William G. Brown," or when the company's regional sales office shows the price of a product as $45.95 and its national sales office shows the same product's price as $43.95. The probability of data inconsistency is greatly reduced in a properly designed database.

- **Improved data access.** The DBMS makes it possible to produce quick answers to ad hoc queries. From a database perspective, a query is a specific request issued to the DBMS for data manipulation—for example, to read or update the data. Simply put, a query is a

question, and an ad hoc query is a spur-of-the-moment question. The DBMS sends back an answer (called the query result set) to the application. For example, end users, when dealing with large amounts of sales data, might want quick answers to questions (ad hoc queries) such as:

- What was the dollar volume of sales by product during the past six months?
- What is the sales bonus figure for each of our salespeople during the past three months?
- How many of our customers have credit balances of $3,000 or more?

- **Improved decision making.** Better-managed data and improved data access make it possible to generate better-quality information, on which better decisions are based. The quality of the information generated depends on the quality of the underlying data. Data quality is a comprehensive approach to promoting the accuracy, validity, and timeliness of the data. While the DBMS does not guarantee data quality, it provides a framework to facilitate data quality initiatives. Data quality concepts will be covered in more detail in Chapter 15, Database Administration and Security.

- **Increased end-user productivity.** The availability of data, combined with the tools that transform data into usable information, empowers end users to make quick, informed decisions that can make the difference between success and failure in the global economy.

The advantages of using a DBMS are not limited to the few just listed. In fact, you will discover many more advantages as you learn more about the technical details of databases and their proper design.

1.3.2 TYPES OF DATABASES

A DBMS can support many different types of databases. Databases can be classified according to the number of users, the database location(s), and the expected type and extent of use. The number of users determines whether the database is classified as single-user or multiuser. A *single-user database* supports only one user at a time. In other words, if user A is using the database, users B and C must wait until user A is done. A single-user database that runs on a personal computer is called a desktop database. In contrast, a *multiuser database* supports multiple users at the same time. When the multiuser database supports a relatively small number of users (usually fewer than 50) or a specific department within an organization, it is called a workgroup database. When the database is used by the entire organization and supports many users (more than 50, usually hundreds) across many departments, the database is known as an enterprise database. Location might also be used to classify the database. For example, a database that supports data located at a single site is called a centralized database. A database that supports data distributed across several different sites is called a distributed database. The extent to which a database can be distributed and the way in which such distribution is managed, *Distributed Database Management Systems*. The most popular way of classifying databases today, however, is based on how they will be used and on the time sensitivity of the information gathered from them. For example, transactions such as product or service sales, payments, and supply purchases reflect critical day-to-day operations. Such transactions must be recorded accurately and immediately. A database that is designed primarily to support a company's dayto-day operations is classified as an *operational database* (sometimes referred to as a *transactional or production database*). In contrast, a data warehouse

focuses primarily on storing data used to generate information required to make tactical or strategic decisions. Such decisions typically require extensive "data massaging" (data manipulation) to extract information to formulate pricing decisions, sales forecasts, market positioning, and so on. Most decision support data are based on data obtained from operational databases over time and stored in data warehouses. Additionally, the data warehouse can store data derived from many sources. To make it easier to retrieve such data, the data warehouse structure is quite different from that of an operational or transactional database.

Databases can also be classified to reflect the degree to which the data are structured. Unstructured data are data that exist in their original (raw) state, that is, in the format in which they were collected. Therefore, unstructured data exist in a format that does not lend itself to the processing that yields information. Structured data are the result of taking unstructured data and formatting (structuring) such data to facilitate storage, use, and the generation of information. You apply structure (format) based on the type of processing that you intend to perform on the data. Some data might not be ready (unstructured) for some types of processing, but they might be ready (structured) for other types of processing. For example, the data value 37890 might refer to a zip code, a sales value, or a product code. If this value represents a zip code or a product code and is stored as text, you cannot perform mathematical computations with it. On the other hand, if this value represents a sales transaction, it is necessary to format it as numeric. To further illustrate the structure concept, imagine a stack of printed paper invoices. If you want to merely store these invoices as images for future retrieval and display, you can scan them and save them in a graphic format. On the other hand, if you want to derive information such as monthly totals and average sales, such graphic storage would not be useful. Instead, you could store the invoice data in a (structured) spreadsheet format so that you can perform the requisite computations. Actually, most data you encounter are best classified as semi-structured. Semistructured data are data that have already been processed to some extent. For example, if you look at a typical Web page, the data are presented to you in a prearranged format to convey some information. The database types mentioned thus far focus on the storage and management of highly structured data. However, corporations are not limited to the use of structured data. They also use semi structured and unstructured data. Just think of the very valuable information that can be found on company e-mails, memos, documents such as procedures and rules, Web pages, and so on. Unstructured and semi structured data storage and management needs are being addressed through a new generation of databases known as XML databases. Extensible Markup Language (XML) is a special language used to represent and manipulate data elements in a textual format. An XML database supports the storage and management of semi structured XML data.

1.4 WHY DATABASE DESIGN IS IMPORTANT

Database design refers to the activities that focus on the design of the database structure that will be used to store and manage end-user data. A database that meets all user requirements does not just happen; its structure must be designed carefully. In fact, database design is such a crucial aspect of working with databases that most of this book is dedicated to the development of good database design techniques. Even a good DBMS will perform poorly with a badly designed database. Proper database design requires the designer to identify precisely the database's expected use. Designing a transactional database emphasizes accurate and consistent data and operational speed. Designing a data warehouse database emphasizes the use of historical and aggregated data. Designing a database to be used in a centralized, single-user environment requires a different approach from that used in the design of a distributed, multiuser database. This book emphasizes

the design of transactional, centralized, single-user, and multiuser databases. Designing appropriate data repositories of integrated information using the twodimensional table structures found in most databases is a process of decomposition. The integrated data must be decomposed properly into its constituent parts, with each part stored in its own table. Further, the relationships between these tables must be carefully considered and implemented so that the integrated view of the data can be re-created later as information for the end user. A well-designed database facilitates data management and generates accurate and valuable information. A poorly designed database is likely to become a breeding ground for difficult-to-trace errors that may lead to bad decision making—and bad decision making can lead to the failure of an organization. Database design is simply too important to be left to luck. That's why college students study database design, why organizations of all types and sizes send personnel to database design seminars, and why database design consultants often make an excellent living.

1.5 EVOLUTION OF FILE SYSTEM DATA PROCESSING

Understanding what a database is, what it does, and the proper way to use it can be clarified by considering what a database is not. A brief explanation of the evolution of file system data processing can be helpful in understanding the data access limitations that databases attempt to overcome. Understanding these limitations is relevant to database designers and developers because database technologies do not make these problems magically disappear—database technologies simply make it easier to create solutions that avoid these problems. Creating database designs that avoid the pitfalls of earlier systems requires that the designer understand what the problems of the earlier systems were and how to avoid them, or else the database technologies are no better than the technologies and techniques that they have replaced.

1.5.1 MANUAL FILE SYSTEMS

In order to be successful, an organization must come up with systems for handling core business tasks. Historically, such systems were often manual, paper-and-pencil systems. The papers within these systems were organized in order to facilitate the expected use of the data. Typically, this was accomplished through a system of file folders and filing cabinets. As long as a data collection was relatively small and an organization's business users had few reporting requirements, the manual system served its role well as a data repository. However, as organizations grew and as reporting requirements became more complex, keeping track of data in a manual file system became more difficult. Therefore, companies looked to computer technology for help.

1.5.2 COMPUTERIZED FILE SYSTEMS

Generating reports from manual file systems was slow and cumbersome. In fact, some business managers faced government-imposed reporting requirements that required weeks of intensive effort each quarter, even when a well-designed manual system was used. Therefore, a data processing (DP) specialist was hired to create a computer-based system that would track data and produce required reports. Initially, the computer files within the file system were similar to the manual files. The description of computer files requires a specialized vocabulary. Every discipline develops its own jargon to enable its practitioners to communicate clearly.

- From the DP specialist's perspective, the computer files within the file system were created to be similar to the manual files. Data management programs were created to add to, update, and delete data from the file.
- From the end user's perspective, the systems separated the users from the data. As the users' competitive environment pushed them to make more and more decisions in less and less time, the delay from when the users conceived of a new way to create information from the data to when the DP specialist could create the programs to generate that information was a source of great frustration.

1.6 PROBLEMS WITH FILE SYSTEM DATA PROCESSING

The file system method of organizing and managing data was a definite improvement over the manual system, and the file system served a useful purpose in data management for over two decades—a very long time in the computer era. Nonetheless, many problems and limitations became evident in this approach. A critique of the file system method serves two major purposes:

- Understanding the shortcomings of the file system enables you to understand the development of modern databases.
- Many of the problems are not unique to file systems. Failure to understand such problems is likely to lead to their duplication in a database environment, even though database technology makes it easy to avoid them.

The following problems associated with file systems, whether created by DP specialists or through a series of spreadsheets, severely challenge the types of information that can be created from the data as well as the accuracy of the information:

- *Lengthy development times.* The first and most glaring problem with the file system approach is that even the simplest data-retrieval task requires extensive programming. With the older file systems, programmers had to specify what must be done and how it was to be done. As you will learn in upcoming chapters, modern databases use a nonprocedural data manipulation language that allows the user to specify what must be done without specifying how it must be done.
- Difficulty of getting quick answers. The need to write programs to produce even the simplest reports makes ad hoc queries impossible. Harried DP specialists who work with mature file systems often receive numerous requests for new reports. They are often forced to say that the report will be ready "next week" or even "next month." If you need the information now, getting it next week or next month will not serve your information needs.
- Complex system administration. System administration becomes more difficult as the number of files in the system expands. Even a simple file system with a few files requires creating and maintaining several file management programs (each file must have its own file management programs that allow the user to add, modify, and delete records; to list the file contents; and to generate reports). Because ad hoc queries are not possible, the file reporting programs can multiply quickly. The problem is compounded by the fact that each department in the organization "owns" its data by creating its own files.
- Lack of security and limited data sharing. Another fault of a file system data repository is a lack of security and limited data sharing. Data sharing and security are closely related. Sharing data among multiple geographically dispersed users introduces a lot of security risks. In terms of spreadsheet data, while many spreadsheet programs provide rudimentary security options, they are not always used, and even when they are used, they are insufficient for robust data sharing among users. In terms of the creation of data management and reporting programs, security and data-sharing features are difficult to

program and are, therefore, often omitted in a file system environment. Such features include effective password protection, the ability to lock out parts of files or parts of the system itself, and other measures designed to safeguard data confidentiality. Even when an attempt is made to improve system and data security, the security devices tend to be limited in scope and effectiveness.

- Extensive programming. Making changes to an existing file structure can be difficult in a file system environment. For example, changing just one field in the original CUSTOMER file would require a program that:

  1. Reads a record from the original file. 2. Transforms the original data to conform to the new structure's storage requirements.
  3. Writes the transformed data into the new file structure.
  4. Repeats steps 2 to 4 for each record in the original file.

In fact, any change to a file structure, no matter how minor, forces modifications in all of the programs that use the data in that file. Modifications are likely to produce errors (bugs), and additional time is spent using a debugging process to find those errors. Those limitations, in turn, lead to problems of structural and data dependence.

1.6.1 STRUCTURAL AND DATA DEPENDENCE

A file system exhibits structural dependence, which means that access to a file is dependent on its structure. For example, adding a customer date-of-birth field to the CUSTOMER file would require the four steps described in the previous section. Given this change, none of the previous programs will work with the new CUSTOMER file structure. Therefore, all of the file system programs must be modified to conform to the new file structure. In short, because the file system application programs are affected by change in the file structure, they exhibit structural dependence. Conversely, structural independence exists when it is possible to make changes in the file structure without affecting the application program's ability to access the data. Even changes in the characteristics of data, such as changing a field from integer to decimal, require changes in all the programs that access the file. Because all data access programs are subject to change when any of the file's data storage characteristics change (that is, changing the data type), the file system is said to exhibit data dependence.

Conversely, data independence exists when it is possible to make changes in the data storage characteristics without affecting the application program's ability to access the data.

The practical significance of data dependence is the difference between the logical data format (how the human being views the data) and the physical data format (how the computer must work with the data). Any program that accesses a file system's file must tell the computer not only what to do but also how to do it. Consequently, each program must contain lines that specify the opening of a specific file type, its record specification, and its field definitions. Data dependence makes the file system extremely cumbersome from the point of view of a programmer and database manager.

1.6.2 DATA REDUNDANCY

The file system's structure makes it difficult to combine data from multiple sources, and its lack of security renders the file system vulnerable to security breaches. The organizational

structure promotes the storage of the same basic data in different locations. (Database professionals use the term islands of information for such scattered data locations.)

The dispersion of data is exacerbated by the use of spreadsheets to store data. In a file system, the entire sales department would share access to the SALES data file through the data management and reporting programs created by the DP specialist. With the use of spreadsheets, it is possible for each member of the sales department to create his or her own copy of the sales data. Because it is unlikely that data stored in different locations will always be updated consistently, the islands of information often contain different versions of the same data. You only need one correct copy of the agent names and phone numbers. Having them occur in more than one place produces data redundancy. *Data redundancy exists when the same data are stored unnecessarily at different places.*

Uncontrolled data redundancy sets the stage for:

- *Poor data security.* Having multiple copies of data increases the chances for a copy of the data to be susceptible to unauthorized access. Database Administration and Security, explores the issues and techniques associated with securing data.
- *Data inconsistency.* Data inconsistency exists when different and conflicting versions of the same data appear in different places. For example, suppose you change an agent's phone number or address in the AGENT file. If you forget to make corresponding changes in the CUSTOMER file, the files contain different data for the same agent. Reports will yield inconsistent results that depend on which version of the data is used.

  Data entry errors are more likely to occur when complex entries (such as 10-digit phone numbers) are made in several different files and/or recur frequently in one or more files. In fact, the CUSTOMER file shown in Figure 1.3 contains just such an entry error: the third record in the CUSTOMER file has a transposed digit in the agent's phone number (615-882-2144 rather than 615-882-1244). It is possible to enter a nonexistent sales agent's name and phone number into the CUSTOMER file, but customers are not likely to be impressed if the insurance agency supplies the name and phone number of an agent who does not exist. Should the personnel manager allow a nonexistent agent to accrue bonuses and benefits? In fact, a data entry error such as an incorrectly spelled name or an incorrect phone number yields the same kind of data integrity problems.

Contents of the CUSTOMER file

C_NAME	C_PHONE	C_ADDRESS	C_ZIP	A_NAME	A_PHONE	TP	AMT	REN
Alfred A. Ramas	615-844-2573	218 Fork Rd., Babs, TN	36123	Leah F. Hahn	615-882-1244	T1	100.00	05-Apr-2010
Leona K. Dunne	713-894-1238	Box 12A, Fox, KY	25246	Alex B. Alby	713-228-1249	T1	250.00	16-Jun-2010
Kathy W. Smith	615-894-2285	125 Oak Ln, Babs, TN	36123	Leah F. Hahn	615-882-2144	S2	150.00	29-Jan-2011
Paul F. Olowski	615-894-2180	217 Lee Ln., Babs, TN	36123	Leah F. Hahn	615-882-1244	S1	300.00	14-Oct-2010
Myron Orlando	615-222-1672	Box 111, New, TN	36155	Alex B. Alby	713-228-1249	T1	100.00	28-Dec-2010
Amy B. O'Brian	713-442-3381	387 Troll Dr., Fox, KY	25246	John T. Okon	615-123-5589	T2	850.00	22-Sep-2010
James G. Brown	615-297-1228	21 Tye Rd., Nash, TN	37118	Leah F. Hahn	615-882-1244	S1	120.00	25-Mar-2011
George Williams	615-290-2556	155 Maple, Nash, TN	37119	John T. Okon	615-123-5589	S1	250.00	17-Jul-2010
Anne G. Farriss	713-382-7185	2119 Elm, Crew, KY	25432	Alex B. Alby	713-228-1249	T2	100.00	03-Dec-2010
Olette K. Smith	615-297-3809	2782 Main, Nash, TN	37118	John T. Okon	615-123-5589	S2	500.00	14-Mar-2011

C_NAME = Customer name
C_PHONE = Customer phone
C_ADDRESS = Customer address
C_ZIP = Customer zip code

A_NAME = Agent name
A_PHONE = Agent phone
TP = Insurance type
AMT = Insurance policy amount, in thousands of $
REN = Insurance renewal date

Contents of the AGENT file

A_NAME	A_PHONE	A_ADDRESS	ZIP	HIRED	YTD_PAY	YTD_FIT	YTD_FICA	YTD_SLS	DEP
Alex B. Alby	713-228-1249	123 Toll, Nash, TN	37119	01-Nov-2000	26566.24	6641.56	2125.30	132737.75	3
Leah F. Hahn	615-882-1244	334 Main, Fox, KY	25246	23-May-1986	32213.78	8053.44	2577.10	138967.35	0
John T. Okon	615-123-5589	452 Elm, New, TN	36155	15-Jun-2005	23198.29	5799.57	1855.86	127093.45	2

A_NAME = Agent name
A_PHONE = Agent phone
A_ADDRESS = Agent address
ZIP = Agent zip code
HIRED = Agent date of hire

YTD_PAY = Year-to-date pay
YTD_FIT = Year-to-date federal income tax paid
YTD_FICA = Year-to-date Social Security taxes paid
YTD_SLS = Year-to-date sales
DEP = Number of dependents

- *Data anomalies.* The dictionary defines anomaly as "an abnormality." Ideally, a field value change should be made in only a single place. Data redundancy, however, fosters an abnormal condition by forcing field value changes in many different locations. Look at the CUSTOMER file. If agent Leah F. Hahn decides to get married and move, the agent name, address, and phone number are likely to change. Instead of making just a single name and/or phone/address change in a single file (AGENT), you must also make the change each time that agent's name, phone number, and address occur in the CUSTOMER file. You could be faced with the prospect of making hundreds of corrections, one for each of the customers served by that agent! The same problem occurs when an agent decides to quit. Each customer served by that agent must be assigned a new agent. Any change in any field value must be correctly made in many places to maintain data integrity. A data anomaly develops when not all of the required changes in the redundant data are made successfully.

The data anomalies are commonly defined as follows:

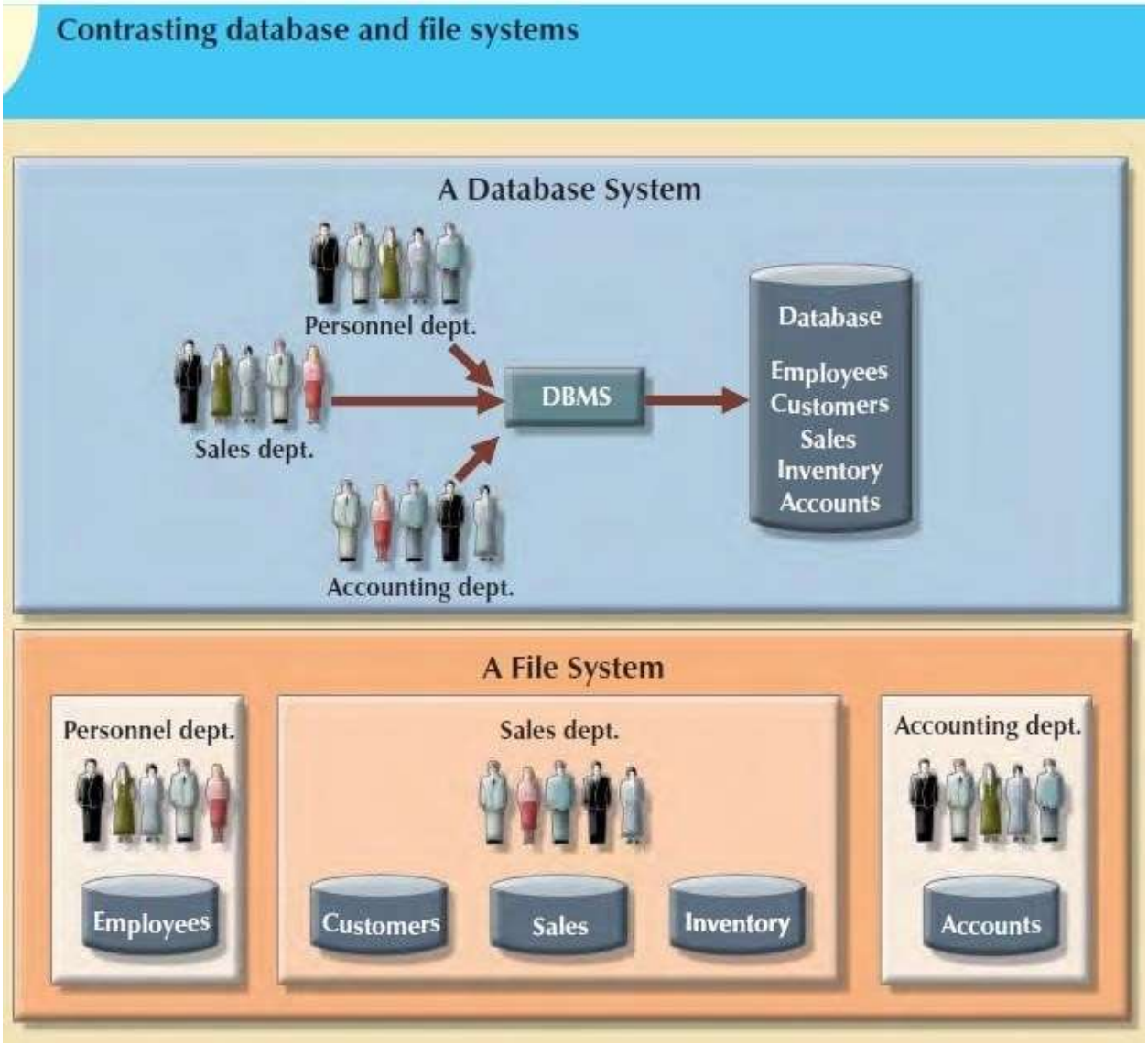
- *Update anomalies.* If agent Leah F. Hahn has a new phone number, that number must be entered in each of the CUSTOMER file records in which Ms. Hahn's phone number is shown. In this case, only three changes must be made. In a large file

system, such a change might occur in hundreds or even thousands of records. Clearly, the potential for data inconsistencies is great.

- *Insertion anomalies.* If only the CUSTOMER file existed, to add a new agent, you would also add a dummy customer data entry to reflect the new agent's addition. Again, the potential for creating data inconsistencies would be great.
- *Deletion anomalies.* If you delete the customers Amy B. O'Brian, George Williams, and Olette K. Smith, you will also delete John T. Okon's agent data. Clearly, this is not desirable.

1.7 DATABASE SYSTEMS

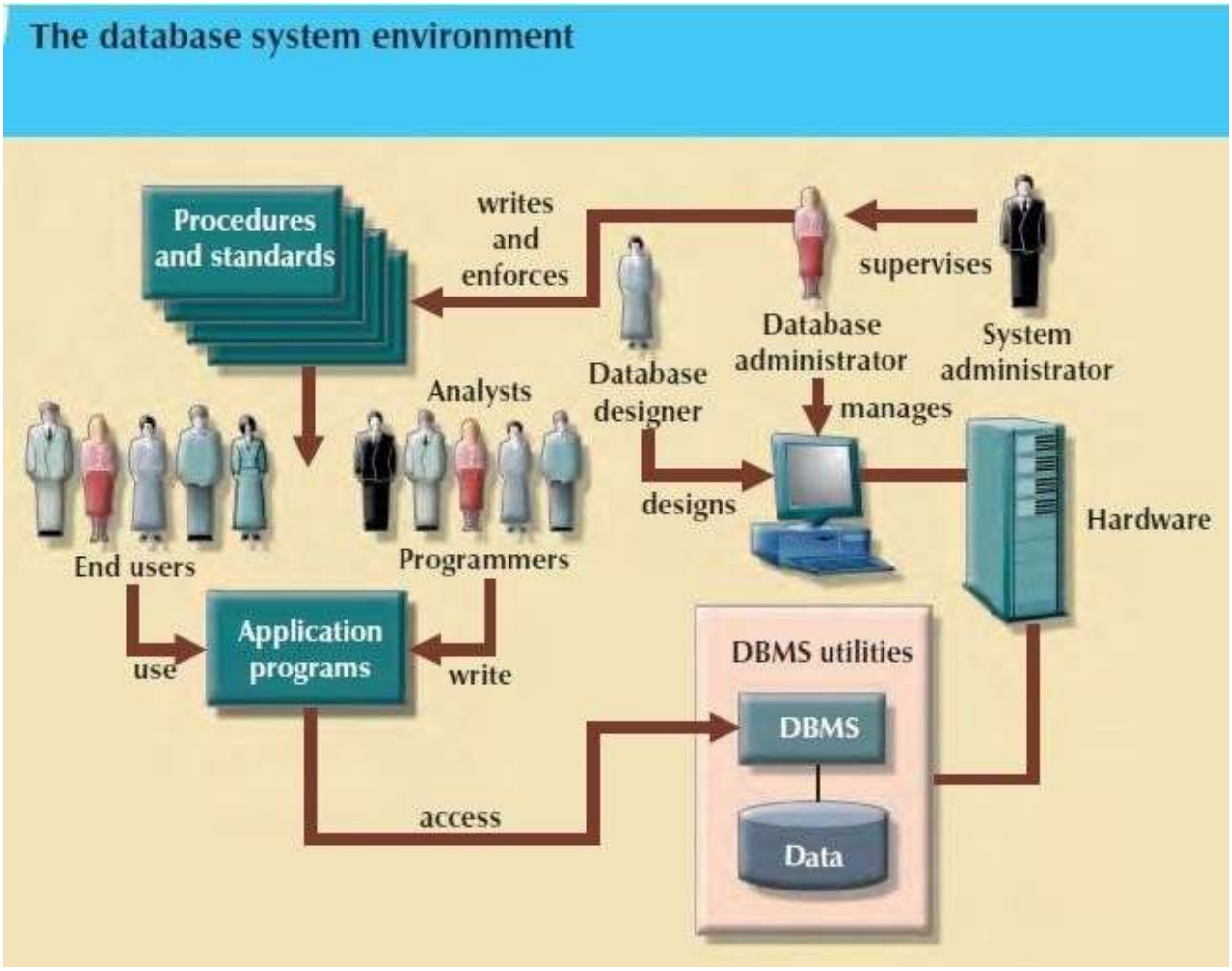
The problems inherent in file systems make using a database system very desirable. Unlike the file system, with its many separate and unrelated files, the database system consists of logically related data stored in a single logical data repository. (The "logical" label reflects the fact that, although the data repository appears to be a single unit to the end user, its contents may actually be physically distributed among multiple data storage facilities and/or locations.) Because the database's data repository is a single logical unit, the database represents a major change in the way end-user data are stored, accessed, and managed. The database's DBMS, shown in the below Figure, provides numerous advantages over file system management, by making it possible to eliminate most of the file system's data inconsistency, data anomaly, data dependence, and structural dependence problems. Better yet, the current generation of DBMS software stores not only the data structures, but also the relationships between those structures and the access paths to those structures—all in a central location. The current generation of DBMS software also takes care of defining, storing, and managing all required access paths to those components.



Remember that the DBMS is just one of several crucial components of a database system. The DBMS may even be referred to as the database system's heart. However, just as it takes more than a heart to make a human being function, it takes more than a DBMS to make a database system function. In the sections that follow, you'll learn what a database system is, what its components are, and how the DBMS fits into the database system picture.

1.9.1 THE DATABASE SYSTEM ENVIRONMENT

The term database system refers to an organization of components that define and regulate the collection, storage, management, and use of data within a database environment. From a general management point of view, the database system is composed of the five major parts shown in below Figure: hardware, software, people, procedures, and data.



Let's take a closer look at the five components shown in Figure above:

- Hardware. Hardware refers to all of the system's physical devices; for example, computers (PCs, workstations, servers, and supercomputers), storage devices, printers, network devices (hubs, switches, routers, fiber optics), and other devices (automated teller machines, ID readers, and so on).

Software. Although the most readily identified software is the DBMS itself, to make the database system function fully, three types of software are needed: operating system software, DBMS software, and application programs and utilities.

- Operating system software manages all hardware components and makes it possible for all other software to run on the computers. Examples of operating system software include Microsoft Windows, Linux, Mac OS, UNIX, and MVS.
- DBMS software manages the database within the database system. Some examples of DBMS software include Microsoft's SQL Server, Oracle Corporation's Oracle, Sun's MySQL, and IBM's DB2.
- Application programs and utility software are used to access and manipulate data in the DBMS and to manage the computer environment in which data access and manipulation take place. Application programs are most commonly used to access data found within the database to generate reports, tabulations, and other information to facilitate decision making. Utilities are the software tools used to help manage the database system's computer components. For example, all of the major DBMS vendors now provide graphical user interfaces (GUIs) to help create database structures, control database access, and monitor database operations.

People. This component includes all users of the database system. On the basis of primary job functions, five types of users can be identified in a database system: system administrators, database administrators, database designers, system analysts and programmers, and end users. Each user type, described below, performs both unique and complementary functions.

- System administrators oversee the database system's general operations.
- Database administrators, also known as DBAs, manage the DBMS and ensure that the database is functioning properly. The DBA's role is sufficiently important to warrant a Database Administration and Security.
- Database designers design the database structure. They are, in effect, the database architects. If the database design is poor, even the best application programmers and the most dedicated DBAs cannot produce a useful database environment. Because organizations strive to optimize their data resources, the database designer's job description has expanded to cover new dimensions and growing responsibilities.
- System analysts and programmers design and implement the application programs. They design and create the data entry screens, reports, and procedures through which end users access and manipulate the database's data.
- End users are the people who use the application programs to run the organization's daily operations. For example, salesclerks, supervisors, managers, and directors are all classified as end users. High-level end users employ the information obtained from the database to make tactical and strategic business decisions.
- Procedures. Procedures are the instructions and rules that govern the design and use of the database system. Procedures are a critical, although occasionally forgotten, component of the system. Procedures play an important role in a company because they enforce the standards by which business is conducted within the organization and with customers. Procedures are also used to ensure that there is an organized way to monitor and audit both the data that enter the database and the information that is generated through the use of those data.
- Data. The word data covers the collection of facts stored in the database. Because data are the raw material from which information is generated, the determination of what data are to be entered into the database and how those data are to be organized is a vital part of the database designer's job.

1.9.2 DBMS FUNCTIONS

A DBMS performs several important functions that guarantee the integrity and consistency of the data in the database. Most of those functions are transparent to end users, and most can be achieved only through the use of a DBMS. They include data dictionary management, data storage management, data transformation and presentation, security management, multiuser access control, backup and recovery management, data integrity management, database access languages and application programming interfaces and database communication interfaces. Each of these functions is explained below.

- Data dictionary management. The DBMS stores definitions of the data elements and their relationships (metadata) in a data dictionary. In turn, all programs that access the data in the database work through the DBMS. The DBMS uses the data dictionary to look up the required data component structures and relationships, thus relieving you from having to code such complex relationships in each program. Additionally, any changes made in a database structure are automatically recorded in the data dictionary, thereby freeing you

from having to modify all of the programs that access the changed structure. In other words, the DBMS provides data abstraction, and it removes structural and data dependence from the system. For example, Figure 1.8 shows how Microsoft SQL Server Express presents the data definition for the CUSTOMER table.

- Data storage management. The DBMS creates and manages the complex structures required for data storage, thus relieving you from the difficult task of defining and programming the physical data characteristics. A modern DBMS provides storage not only for the data, but also for related data entry forms or screen definitions, report definitions, data validation rules, procedural code, structures to handle video and picture formats, and so on. Data storage management is also important for database performance tuning.

Performance tuning relates to the activities that make the database perform more efficiently in terms of storage and access speed. Although the user sees the database as a single data storage unit, the DBMS actually stores the database in multiple physical data files. Such data files may even be stored on different storage media. Therefore, the DBMS doesn't have to wait for one disk request to finish before the next one starts. In other words, the DBMS can fulfill database requests concurrently.

Data transformation and presentation. The DBMS transforms entered data to conform to required data structures. The DBMS relieves you of the chore of making a distinction between the logical data format and the physical data format. That is, the DBMS formats the physically retrieved data to make it conform to the user's logical expectations. For example, imagine an enterprise database used by a multinational company. An end user in England would expect to enter data such as July 11, 2010, as "11/07/2010." In contrast, the same date would be entered in the United States as "07/11/2010." Regardless of the data presentation format, the DBMS must manage the date in the proper format for each country.

- Security management. The DBMS creates a security system that enforces user security and data privacy. Security rules determine which users can access the database, which data items each user can access, and which data operations (read, add, delete, or modify) the user can perform. This is especially important in multiuser database systems. All database users may be authenticated to the DBMS through a username and password or through biometric authentication such as a fingerprint scan. The DBMS uses this information to assign access privileges to various database components such as queries and reports.
- Multiuser access control. To provide data integrity and data consistency, the DBMS uses sophisticated algorithms to ensure that multiple users can access the database concurrently without compromising the integrity of the database.
- Backup and recovery management. The DBMS provides backup and data recovery to ensure data safety and integrity. Current DBMS systems provide special utilities that allow the DBA to perform routine and special backup and restore procedures. Recovery management deals with the recovery of the database after a failure, such as a bad sector in the disk or a power failure. Such capability is critical to preserving the database's integrity.
- Data integrity management. The DBMS promotes and enforces integrity rules, thus minimizing data redundancy and maximizing data consistency. The data relationships stored in the data dictionary are used to enforce data integrity. Ensuring data integrity is especially important in transaction-oriented database systems.
- Database access languages and application programming interfaces. The DBMS provides data access through a query language. A query language is a nonprocedural language—one that lets the user specify what must be done without having to specify how it is to be done. Structured Query Language (SQL) is the de facto query language and data access standard supported by the majority of DBMS vendors. The DBMS also provides application programming interfaces to procedural languages such as COBOL, C, Java,

Visual Basic.NET, and C#. In addition, the DBMS provides administrative utilities used by the DBA and the database designer to create, implement, monitor, and maintain the database.

- Database communication interfaces. Current-generation DBMSs accept end-user requests via multiple, different network environments. For example, the DBMS might provide access to the database via the Internet through the use of Web browsers such as Mozilla Firefox or Microsoft Internet Explorer. In this environment, communications can be accomplished in several ways:
- End users can generate answers to queries by filling in screen forms through their preferred Web browser.
- The DBMS can automatically publish predefined reports on a Website.
- The DBMS can connect to third-party systems to distribute information via e-mail or other productivity applications.

1.9.3 MANAGING THE DATABASE SYSTEM: A SHIFT IN FOCUS

The introduction of a database system over the file system provides a framework in which strict procedures and standards can be enforced. Consequently, the role of the human component changes from an emphasis on programming (in the file system) to a focus on the broader aspects of managing the organization's data resources and on the administration of the complex database software itself. The database system makes it possible to tackle far more sophisticated uses of the data resources, as long as the database is designed to make use of that available power. The kinds of data structures created within the database and the extent of the relationships among them play a powerful role in determining the effectiveness of the database system. Although the database system yields considerable advantages over previous data management approaches, database systems do carry significant disadvantages. For example:

- *Increased costs.* Database systems require sophisticated hardware and software and highly skilled personnel. The cost of maintaining the hardware, software, and personnel required to operate and manage a database system can be substantial. Training, licensing, and regulation compliance costs are often overlooked when database systems are implemented.
- *Management complexity.* Database systems interface with many different technologies and have a significant impact on a company's resources and culture. The changes introduced by the adoption of a database system must be properly managed to ensure that they help advance the company's objectives. Given the fact that database systems hold crucial company data that are accessed from multiple sources, security issues must be assessed constantly.
- Maintaining currency. To maximize the efficiency of the database system, you must keep your system current. Therefore, you must perform frequent updates and apply the latest patches and security measures to all components. Because database technology advances rapidly, personnel training costs tend to be significant.
- Vendor dependence. Given the heavy investment in technology and personnel training, companies might be reluctant to change database vendors. As a consequence, vendors are less likely to offer pricing point advantages to existing customers, and those customers might be limited in their choice of database system components.
- Frequent upgrade/replacement cycles. DBMS vendors frequently upgrade their products by adding new functionality. Such new features often come bundled in new upgrade versions of the software. Some of these versions require hardware upgrades. Not only do the upgrades themselves cost money, but it also costs money to train database users and

administrators to properly use and manage the new features. Now that we have considered what a database and DBMS are, and why they are necessary, it is natural for our thoughts to turn to developing the skills of database design. However, before we can create a design, we must know what tools are at our disposal. Throughout this chapter, we have generalized the discussion of database technology such that it appears that there is a single, common approach to database design. As a database designer and developer, however, you need to understand that there are different approaches, and you need to know how these approaches influence the designs that you can create and how those designs are modeled.

SUMMARY:

- Data are raw facts. Information is the result of processing data to reveal its meaning. Accurate, relevant, and timely information is the key to good decision making, and good decision making is the key to organizational survival in a global environment.
- Data are usually stored in a database. To implement a database and to manage its contents, you need a database management system (DBMS). The DBMS serves as the intermediary between the user and the database. The database contains the data you have collected and "data about data," known as metadata.
- Database design defines the database structure. A well-designed database facilitates data management and generates accurate and valuable information. A poorly designed database can lead to bad decision making, and bad decision making can lead to the failure of an organization.
- Databases evolved from manual and then computerized file systems. In a file system, data are stored in independent files, each requiring its own data management programs. Although this method of data management is largely outmoded, understanding its characteristics makes database design easier to comprehend.
- Some limitations of file system data management are that it requires extensive programming, system administration can be complex and difficult, making changes to existing structures is difficult, and security features are likely to be inadequate. Also, independent files tend to contain redundant data, leading to problems of structural and data dependence.
- Database management systems were developed to address the file system's inherent weaknesses. Rather than depositing data in independent files, a DBMS presents the database to the end user as a single data repository. This arrangement promotes data sharing, thus eliminating the potential problem of islands of information. In addition, the DBMS enforces data integrity, eliminates redundancy, and promotes data security.

REVIEW QUESTIONS:

1. Define each of the following terms:
a. data
b. field
c. record
d. file
2. What is data redundancy, and which characteristics of the file system can lead to it?

3. What is data independence, and why is it lacking in file systems?
4. What is a DBMS, and what are its functions?
5. What is structural independence, and why is it important?
9. Explain the difference between data and information.
9. What is the role of a DBMS, and what are its advantages? What are its disadvantages?
7. List and describe the different types of databases.
8. What are the main components of a database system?
10. What are metadata?
11. Explain why database design is important.
12. What are the potential costs of implementing a database system?
13. Use examples to compare and contrast unstructured and structured data. Which type is more prevalent in a typical business environment?
14. What are some basic database functions that a spreadsheet cannot perform?

Chapter 2: Data Models

This chapter examines data modeling. Data modeling is the first step in the database design journey, serving as a bridge between real-world objects and the database that resides in the computer.

One of the most vexing problems of database design is that designers, programmers, and end users see data in different ways. Consequently, different views of the same data can lead to database designs that do not reflect an organization's actual operation, thus failing to meet enduser needs and data efficiency requirements. To avoid such failures, database designers must obtain a precise description of the nature of the data and of the many uses of that data within the organization. Communication among database designers, programmers, and end users should be frequent and clear. Data modeling clarifies such communication by reducing the complexities of database design to more easily understood abstractions that define entities and the relations among them. First, you will learn what some of the basic data-modeling concepts are and how current data models developed from earlier models. Tracing the development of those database models will help you understand the database design and implementation issues that are addressed in the rest of this book. Second, you will be introduced to the entity relationship diagram (ERD) as a data-modeling tool. ER diagrams can be drawn using a variety of notations. Within this chapter, you will be introduced to the traditional Chen notation, the more current Crow's Foot notation, and the newer class diagram notation, which is part of the Unified Modeling Language (UML). Finally, you will learn how various degrees of data abstraction help reconcile varying views of the same data.

2.1 DATA MODELING AND DATA MODELS

Database design focuses on how the database structure will be used to store and manage end-user data. Data modeling, the first step in designing a database, refers to the process of creating a specific data model for a determined problem domain. (A problem domain is a clearly defined area within the real-world environment, with well-defined scope and boundaries, that is to be systematically addressed.) A data model is a relatively simple representation, usually graphical, of more complex real-world data structures. In general terms, a model is an abstraction of a more

complex real-world object or event. A model's main function is to help you understand the complexities of the real-world environment. Within the database environment, a data model represents data structures and their characteristics, relations, constraints, transformations, and other constructs with the purpose of supporting a specific problem domain. Data modeling is an iterative, progressive process. You start with a simple understanding of the problem domain, and as your understanding of the problem domain increases, so does the level of detail of the data model. Done properly, the final data model is in effect a "blueprint" containing all the instructions to build a database that will meet all end-user requirements. This blueprint is narrative and graphical in nature, meaning that it contains both text descriptions in plain, unambiguous language and clear, useful diagrams depicting the main data elements.

2.2 THE IMPORTANCE OF DATA MODELS

Data models can facilitate interaction among the designer, the applications programmer, and the end user. A well-developed data model can even foster improved understanding of the organization for which the database design is developed. In short, data models are a communication tool. This important aspect of data modeling was summed up neatly by a client whose reaction was as follows: "I created this business, I worked with this business for years, and this is the first time I've really understood how all the pieces really fit together." The importance of data modeling cannot be overstated. Data constitute the most basic information units employed by a system. Applications are created to manage data and to help transform data into information. But data are viewed in different ways by different people. For example, contrast the (data) view of a company manager with that of a company clerk. Although the manager and the clerk both work for the same company, the manager is more likely to have an enterprise-wide view of company data than the clerk. Even different managers view data differently. For example, a company president is likely to take a universal view of the data because he or she must be able to tie the company's divisions to a common (database) vision. A purchasing manager in the same company is likely to have a more restricted view of the data, as is the company's inventory manager. In effect, each department manager works with a subset of the company's data. The inventory manager is more concerned about inventory levels, while the purchasing manager is more concerned about the cost of items and about personal/business relationships with the suppliers of those items.

Applications programmers have yet another view of data, being more concerned with data location, formatting, and specific reporting requirements. Basically, applications programmers translate company policies and procedures from a variety of sources into appropriate interfaces, reports, and query screens. The different users and producers of data and information often reflect the "blind people and the elephant" analogy: the blind person who felt the elephant's trunk had quite a different view of the elephant from the one who felt the elephant's leg or tail. What is needed is a view of the whole elephant. Similarly, a house is not a random collection of rooms; if someone is going to build a house, he or she should first have the overall view that is provided by blueprints. Likewise, a sound data environment requires an overall database blueprint based on an appropriate data model. When a good database blueprint is available, it does not matter that an applications programmer's view of the data is different from that of the manager and/or the end user. Conversely, when a good database blueprint is not available, problems are likely to ensue. For instance, an inventory management program and an order entry system may use conflicting product-numbering schemes, thereby costing the company thousands (or even millions) of dollars. Keep in mind that a house blueprint is an abstraction; you cannot live in the blueprint. Similarly, the data model is an abstraction; you cannot draw the required data out of the data model. Just as

you are not likely to build a good house without a blueprint, you are equally unlikely to create a good database without first creating an appropriate data model.

2.3 DATA MODEL BASIC BUILDING BLOCKS

The basic building blocks of all data models are entities, attributes, relationships, and constraints. An entity is anything (a person, a place, a thing, or an event) about which data are to be collected and stored. An entity represents a particular type of object in the real world. Because an entity represents a particular type of object, entities are "distinguishable"—that is, each entity occurrence is unique and distinct. For example, a CUSTOMER entity would have many distinguishable customer occurrences, such as John Smith, Pedro Dinamita, Tom Strickland, etc. Entities may be physical objects, such as customers or products, but entities may also be abstractions, such as flight routes or musical concerts. An attribute is a characteristic of an entity. For example, a CUSTOMER entity would be described by attributes such as customer last name, customer first name, customer phone, customer address, and customer credit limit. Attributes are the equivalent of fields in file systems. A relationship describes an association among entities. For example, a relationship exists between customers and agents that can be described as follows: an agent can serve many customers, and each customer may be served by one agent. Data models use three types of relationships: one-to-many, many-to-many, and one-to-one. Database designers usually use the shorthand notations 1:M or 1..*, M:N or *..*, and 1:1 or 1..1, respectively. (Although the M:N notation is a standard label for the many-to-many relationship, the label M:M may also be used.) The following examples illustrate the distinctions among the three. One-to-many (1:M or 1..*) relationship. A painter paints many different paintings, but each one of them is painted by only one painter. Thus, the painter (the "one") is related to the paintings (the "many"). Therefore, database designers label the relationship "PAINTER paints PAINTING" as 1:M. (Note that entity names are often capitalized as a convention, so they are easily identified.) Similarly, a customer (the "one") may generate many invoices, but each invoice (the "many") is generated by only a single customer. The "CUSTOMER generates INVOICE" relationship would also be labeled 1:M. _ Many-to-many (M:N or *..*) relationship. An employee may learn many job skills, and each job skill may be learned by many employees.

Database designers label the relationship "EMPLOYEE learns SKILL" as M:N. Similarly, a student can take many classes and each class can be taken by many students, thus yielding the M:N relationship label for the relationship expressed by "STUDENT takes CLASS."

- One-to-one (1:1 or 1..1) relationship. A retail company's management structure may require that each of its stores be managed by a single employee. In turn, each store manager, who is an employee, manages only a single store. Therefore, the relationship "EMPLOYEE manages STORE" is labeled 1:1. The preceding discussion identified each relationship in both directions; that is, relationships are bidirectional:
- One CUSTOMER can generate many INVOICEs.
- Each of the many INVOICEs is generated by only one CUSTOMER. A constraint is a restriction placed on the data. Constraints are important because they help to ensure data integrity. Constraints are normally expressed in the form of rules. For example:
- An employee's salary must have values that are between 6,000 and 350,000.
- A student's GPA must be between 0.00 and 4.00.
- Each class must have one and only one teacher. How do you properly identify entities, attributes, relationships, and constraints? The first step is to clearly identify the business rules for the problem domain you are modeling.

2.4 BUSINESS RULES

When database designers go about selecting or determining the entities, attributes, and relationships that will be used to build a data model, they might start by gaining a thorough understanding of what types of data are in an organization, how the data are used, and in what time frames they are used. But such data and information do not, by themselves, yield the required understanding of the total business. From a database point of view, the collection of data becomes meaningful only when it reflects properly defined business rules. A business rule is a brief, precise, and unambiguous description of a policy, procedure, or principle within a specific organization. In a sense, business rules are misnamed: they apply to any organization, large or small—a business, a government unit, a religious group, or a research laboratory—that stores and uses data to generate information. Business rules, derived from a detailed description of an organization's operations, help to create and enforce actions within that organization's environment. Business rules must be rendered in writing and updated to reflect any change in the organization's operational environment. Properly written business rules are used to define entities, attributes, relationships, and constraints. Any time you see relationship statements such as "an agent can serve many customers, and each customer can be served by only one agent," you are seeing business rules at work. You will see the application of business rules throughout this book, especially in the chapters devoted to data modeling and database design. To be effective, business rules must be easy to understand and widely disseminated, to ensure that every person in the organization shares a common interpretation of the rules. Business rules describe, in simple language, the main and distinguishing characteristics of the data as viewed by the company. Examples of business rules are as follows:

- A customer may generate many invoices.
- An invoice is generated by only one customer.
- A training session cannot be scheduled for fewer than 10 employees or for more than 30 employees.

Note that those business rules establish entities, relationships, and constraints. For example, the first two business rules establish two entities (CUSTOMER and INVOICE) and a 1:M relationship between those two entities. The third business rule establishes a constraint (no fewer than 10 people and no more than 30 people), two entities (EMPLOYEE and TRAINING), and a relationship between EMPLOYEE and TRAINING.

2.4.1 DISCOVERING BUSINESS RULES

The main sources of business rules are company managers, policy makers, department managers, and written documentation such as a company's procedures, standards, and operations manuals. A faster and more direct source of business rules is direct interviews with end users. Unfortunately, because perceptions differ, end users are sometimes a less reliable source when it comes to specifying business rules. For example, a maintenance department mechanic might believe that any mechanic can initiate a maintenance procedure, when actually only mechanics with inspection authorization can perform such a task. Such a distinction might seem trivial, but it can have major legal consequences. Although end users are crucial contributors to the development of business rules, it pays to verify end-user perceptions. Too often, interviews with several people who perform the same job yield very different perceptions of what the job components are. While such a discovery may point to "management problems," that general diagnosis does not help the database designer. The database designer's job is to reconcile such differences and verify the results of the

reconciliation to ensure that the business rules are appropriate and accurate. The process of identifying and documenting business rules is essential to database design for several reasons:

- They help to standardize the company's view of data.
- They can be a communications tool between users and designers.
- They allow the designer to understand the nature, role, and scope of the data.
- They allow the designer to understand business processes.
- They allow the designer to develop appropriate relationship participation rules and constraints and to create an accurate data model. Of course, not all business rules can be modeled. For example, a business rule that specifies that "no pilot can fly more

than 10 hours within any 24-hour period" cannot be modeled. However, such a business rule can be enforced by application software.

2.4.2 TRANSLATING BUSINESS RULES INTO DATA MODEL COMPONENTS

Business rules set the stage for the proper identification of entities, attributes, relationships, and constraints. In the real world, names are used to identify objects. If the business environment wants to keep track of the objects, there will be specific business rules for them. As a general rule, a noun in a business rule will translate into an entity in the model, and a verb (active or passive) associating nouns will translate into a relationship among the entities. For example, the business rule "a customer may generate many invoices" contains two nouns (customer and invoices) and a verb (generate) that associates the nouns. From this business rule, you could deduce that:

- Customer and invoice are objects of interest for the environment and should be represented by their respective entities.
- There is a "generate" relationship between customer and invoice.

To properly identify the type of relationship, you should consider that relationships are bidirectional; that is, they go both ways. For example, the business rule "a customer may generate many invoices" is complemented by the business rule "an invoice is generated by only one customer." In that case, the relationship is one-to-many (1:M). Customer is the "1" side, and invoice is the "many" side. As a general rule, to properly identify the relationship type, you should ask two questions:

- How many instances of B are related to one instance of A?
- How many instances of A are related to one instance of B? For example, you can assess the relationship between student and class by asking two questions:
- In how many classes can one student enroll? Answer: many classes.
- How many students can enroll in one class? Answer: many students. Therefore, the relationship between student and class is many-to-many (M:N). You will have many opportunities to determine the relationships between entities as you proceed through this book, and soon the process will become second nature.

2.4.3 NAMING CONVENTIONS

During the translation of business rules to data model components, you identify entities, attributes, relationships, and constraints. This identification process includes naming the object in a way that makes the object unique and distinguishable from other objects in the problem domain. Therefore, it is important that you pay special attention to how you name the objects you are discovering. Entity names should be descriptive of the objects in the business environment, and use terminology that is familiar to the users. An attribute name should also be descriptive of the data represented

by that attribute. It is also a good practice to prefix the name of an attribute with the name of the entity (or an abbreviation of the entity name) in which it occurs. For example, in the CUSTOMER entity, the customer's credit limit may be called CUS_CREDIT_LIMIT. The CUS indicates that the attribute is descriptive of the CUSTOMER entity, while CREDIT_LIMIT makes it easy to recognize the data that will be contained in the attribute. This will become increasingly important in later chapters when we discuss the need to use common attributes to specify relationships between entities. The use of a proper naming convention will improve the data model's ability to facilitate communication among the designer, application programmer, and the end user. In fact, a proper naming convention can go a long way toward making your model self-documenting.

2.5 THE EVOLUTION OF DATA MODELS

The quest for better data management has led to several models that attempt to resolve the file system's critical shortcomings. These models represent schools of thought as to what a database is, what it should do, the types of structures that it should employ, and the technology that would be used to implement these structures. Perhaps confusingly, these models are called data models just as are the graphical data models that we have been discussing. This section gives an overview of the major data models in roughly chronological order. You will discover that many of the "new" database concepts and structures bear a remarkable resemblance to some of the "old" data model concepts and structures. Table below traces the evolution of the major data models.

Evolution of Major Data Models

GENERATION	TIME	DATA MODEL	EXAMPLES	COMMENTS
First	1960s–1970s	File system	VMS/VSAM	Used mainly on IBM mainframe systems Managed records, not relationships
Second	1970s	Hierarchical and network	IMS ADABAS IDS-II	Early database systems Navigational access
Third	Mid-1970s to present	Relational	DB2 Oracle MS SQL-Server MySQL	Conceptual simplicity Entity relationship (ER) modeling and support for relational data modeling
Fourth	Mid-1980s to present	Object-oriented Object/relational (O/R)	Versant Objectivity/DB DB/2 UDB Oracle 11g	Object/relational supports object data types Star Schema support for data warehousing Web databases become common
Next generation	Present to future	XML Hybrid DBMS	dbXML Tamino DB2 UDB Oracle 11g MS SQL Server	Unstructured data support O/R model supports XML documents Hybrid DBMS adds an object front end to relational databases

2.5.1 HIERARCHICAL AND NETWORK MODELS

The hierarchical model was developed in the 1960s to manage large amounts of data for complex manufacturing projects such as the Apollo rocket that landed on the moon in 1968. Its basic logical structure is represented by an upside-down tree. The hierarchical structure contains levels, or segments. A segment is the equivalent of a file system's record type. Within the hierarchy, a higher layer is perceived as the parent of the segment directly beneath it, which is called the child. The

hierarchical model depicts a set of one-to-many (1:M) relationships between a parent and its children segments. (Each parent can have many children, but each child has only one parent.)
The network model was created to represent complex data relationships more effectively than the hierarchical model, to improve database performance, and to impose a database standard. In the network model, the user perceives the network database as a collection of records in 1:M relationships. However, unlike the hierarchical model, the network model allows a record to have more than one parent. While the network database model is generally not used today, the definitions of standard database concepts that emerged with the network model are still used by modern data models. Some important concepts that were defined at this time are:

- The schema, which is the conceptual organization of the entire database as viewed by the database administrator.
- The subschema, which defines the portion of the database "seen" by the application programs that actually produce the desired information from the data contained within the database.
- A data management language (DML), which defines the environment in which data can be managed and to work with the data in the database.
- A schema data definition language (DDL), which enables the database administrator to define the schema components.

As information needs grew and as more sophisticated databases and applications were required, the network model became too cumbersome. The lack of ad hoc query capability put heavy pressure on programmers to generate the code required to produce even the simplest reports. And although the existing databases provided limited data independence, any structural change in the database could still produce havoc in all application programs that drew data from the database. Because of the disadvantages of the hierarchical and network models, they were largely replaced by the relational data model in the 1980s.

2.5.2 THE RELATIONAL MODEL

The relational model was introduced in 1970 by E. F. Codd (of IBM) in his landmark paper "A Relational Model of Data for Large Shared Databanks" (Communications of the ACM, June 1970, pp. 377−387). The relational model represented a major breakthrough for both users and designers. To use an analogy, the relational model produced an "automatic transmission" database to replace the "standard transmission" databases that preceded it. Its conceptual simplicity set the stage for a genuine database revolution. The relational model foundation is a mathematical concept known as a relation. To avoid the complexity of abstract mathematical theory, you can think of a relation (sometimes called a table) as a matrix composed of intersecting rows and columns. Each row in a relation is called a tuple. Each column represents an attribute. The relational model also describes a precise set of data manipulation constructs based on advanced mathematical concepts. In 1970, Codd's work was considered ingenious but impractical. The relational model's conceptual simplicity was bought at the expense of computer overhead; computers at that time lacked the power to implement the relational model. Fortunately, computer power grew exponentially, as did operating system efficiency. Better yet, the cost of computers diminished rapidly as their power grew. Today even PCs, costing a fraction of what their mainframe ancestors did, can run sophisticated relational database software such as Oracle, DB2, Microsoft SQL Server, MySQL, and other mainframe relational software. The relational data model is implemented through a very sophisticated relational database management system (RDBMS). The RDBMS performs the same basic functions provided by the hierarchical and network DBMS systems, in addition to a host of other functions that make the relational data model easier to understand and implement.

Arguably the most important advantage of the RDBMS is its ability to hide the complexities of the relational model from the user. The RDBMS manages all of the physical details, while the user sees the relational database as a collection of tables in which data are stored. The user can manipulate and query the data in a way that seems intuitive and logical. Tables are related to each other through the sharing of a common attribute (value in a column). For example, the CUSTOMER table contain a sales agent's number that is also contained in the AGENT table.

2.5.3 THE ENTITY RELATIONSHIP MODEL

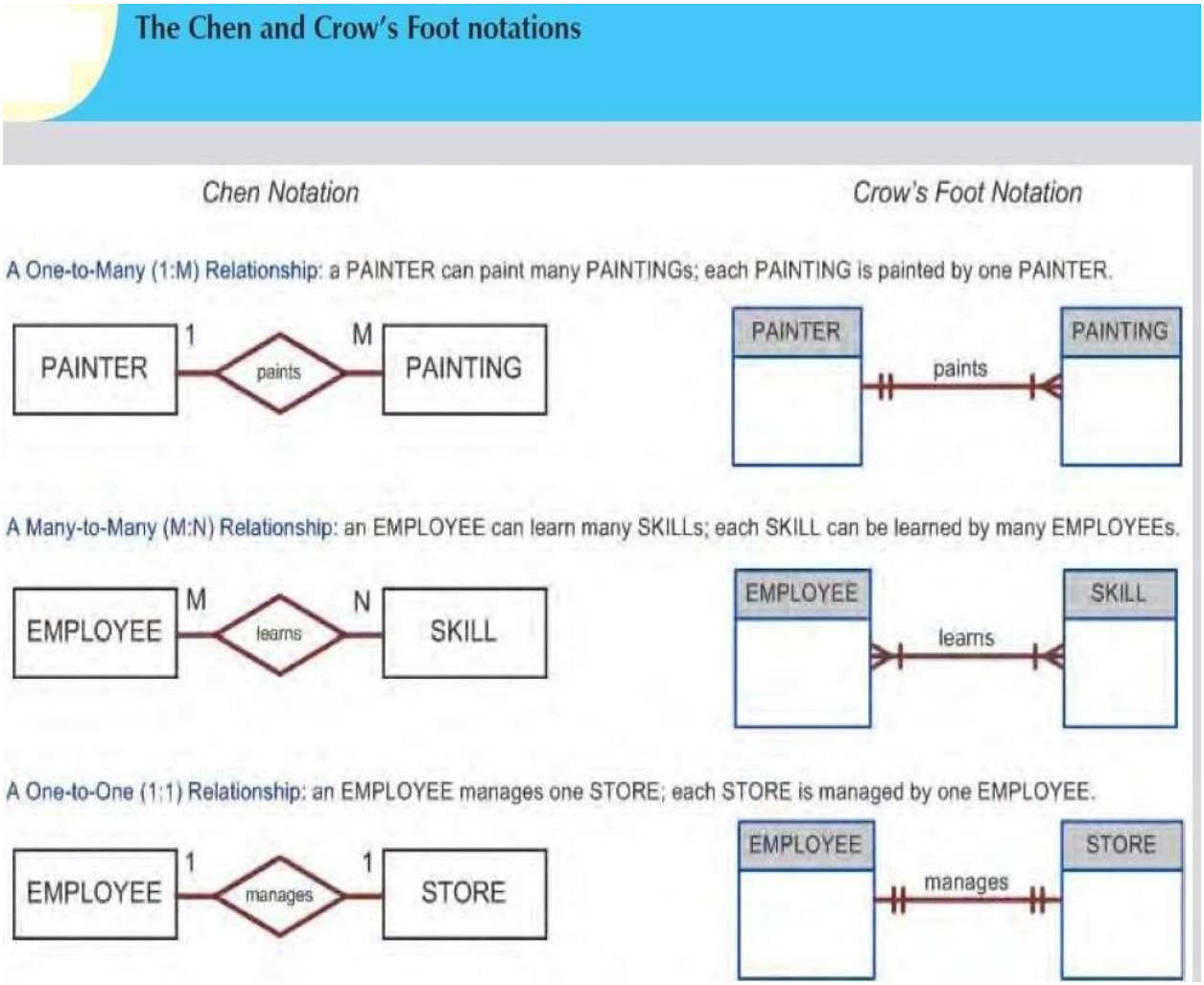
The conceptual simplicity of relational database technology triggered the demand for RDBMSs. In turn, the rapidly increasing requirements for transaction and information created the need for more complex database implementation structures, thus creating the need for more effective database design tools. (Building a skyscraper requires more detailed design activities than building a doghouse, for example.) Complex design activities require conceptual simplicity to yield successful results. Although the relational model was a vast improvement over the hierarchical and network models, it still lacked the features that would make it an effective database design tool. Because it is easier to examine structures graphically than to describe them in text, database designers prefer to use a graphical tool in which entities and their relationships are pictured. Thus, the entity relationship (ER) model, or ERM, has become a widely accepted standard for data modeling. Peter Chen first introduced the ER data model in 1976; it was the graphical representation of entities and their relationships in a database structure that quickly became popular because it complemented the relational data model concepts. The relational data model and ERM combined to provide the foundation for tightly structured database design. ER models are normally represented in an entity relationship diagram (ERD), which uses graphical representations to model database components. The ER model is based on the following components:

_ Entity. Earlier in this chapter, an entity was defined as anything about which data are to be collected and stored. An entity is represented in the ERD by a rectangle, also known as an entity box. The name of the entity, a noun, is written in the center of the rectangle. The entity name is generally written in capital letters and is written in the singular form: PAINTER rather than PAINTERS, and EMPLOYEE rather than EMPLOYEES. Usually, when applying the ERD to the relational model, an entity is mapped to a relational table. Each row in the relational table is known as an entity instance or entity occurrence in the ER model. Each entity is described by a set of attributes that describes particular characteristics of the entity. For example, the entity EMPLOYEE will have attributes such as a Social Security number, a last name, and a first name. (Chapter 4 explains how attributes are included in the ERD.)

_ Relationships. Relationships describe associations among data. Most relationships describe associations between two entities. When the basic data model components were introduced, three types of relationships among data were illustrated: one-to-many (1:M), many-to-many (M:N), and one-to-one (1:1). The ER model uses the term connectivity to label the relationship types. The name of the relationship is usually an active or passive verb. For example, a PAINTER paints many PAINTINGs; an EMPLOYEE learns many SKILLs; an EMPLOYEE manages a STORE. Figure shows the different types of relationships using two ER notations: the original Chen notation and the more current Crow's Foot notation. The left side of the ER diagram shows the Chen notation, based on Peter Chen's landmark paper. In this notation, the connectivity is written next to each entity box. Relationships are represented by a diamond connected to the related entities through a relationship line. The relationship name is written inside the diamond.

The right side of below Figure illustrates the Crow's Foot notation. The name "Crow's Foot" is derived from the three-pronged symbol used to represent the "many" side of the relationship. As you examine the basic Crow's Foot ERD in Figur, note that the connectivities are represented by

symbols. For example, the "1" is represented by a short line segment, and the "M" is represented by the three-pronged "crow's foot." In this example, the relationship name is written above the relationship line. In Figure, entities and relationships are shown in a horizontal format, but they may also be oriented vertically. The entity location and the order in which the entities are presented are immaterial; just remember to read a 1:M relationship from the "1" side to the "M" side. The Crow's Foot notation is used as the design standard in this book. However, the Chen notation is used to illustrate some of the ER modeling concepts whenever necessary. Most data modeling tools let you select the Crow's Foot notation. Microsoft Visio Professional software was used to generate the Crow's Foot designs you will see in subsequent chapters. Its exceptional visual simplicity makes the ER model the dominant database modeling and design tool. Nevertheless, the search for better data-modeling tools continues as the data environment continues to evolve.



2.5.4 THE OBJECT-ORIENTED (OO) MODEL

Increasingly complex real-world problems demonstrated a need for a data model that more closely represented the real world. In the object-oriented data model (OODM), both data and their relationships are contained in a single structure known as an object. In turn, the OODM is the basis for the object-oriented database management system (OODBMS).

An OODM reflects a very different way to define and use entities. Like the relational model's entity, an object is described by its factual content. But quite unlike an entity, an object includes information about relationships between the facts within the object, as well as information about

its relationships with other objects. Therefore, the facts within the object are given greater meaning. The OODM is said to be a semantic data model because semantic indicates meaning. Subsequent OODM development has allowed an object to also contain all operations that can be performed on it, such as changing its data values, finding a specific data value, and printing data values. Because objects include data, various types of relationships, and operational procedures, the object becomes self-contained, thus making the object—at least potentially—a basic building block for autonomous structures.

The OO data model is based on the following components:

- An object is an abstraction of a real-world entity. In general terms, an object may be considered equivalent to an ER model's entity. More precisely, an object represents only one occurrence of an entity. (The object's semantic content is defined through several of the items in this list.)
- Attributes describe the properties of an object. For example, a PERSON object includes the attributes Name, Social Security Number, and Date of Birth.
- Objects that share similar characteristics are grouped in classes. A class is a collection of similar objects with shared structure (attributes) and behavior (methods). In a general sense, a class resembles the ER model's entity set. However, a class is different from an entity set in that it contains a set of procedures known as methods. A class's method represents a real-world action such as finding a selected PERSON's name, changing a PERSON's name, or printing a PERSON's address. In other words, methods are the equivalent of procedures in traditional programming languages. In OO terms, methods define an object's behavior.
- Classes are organized in a class hierarchy. The class hierarchy resembles an upside-down tree in which each class has only one parent. For example, the CUSTOMER class and the EMPLOYEE class share a parent PERSON class. (Note the similarity to the hierarchical data model in this respect.)
- Inheritance is the ability of an object within the class hierarchy to inherit the attributes and methods of the classes above it. For example, two classes, CUSTOMER and EMPLOYEE, can be created as subclasses from the class PERSON. In this case, CUSTOMER and EMPLOYEE will inherit all attributes and methods from PERSON.

Object-oriented data models are typically depicted using Unified Modeling Language (UML) class diagrams. Unified Modeling Language (UML) is a language based on OO concepts that describes a set of diagrams and symbols that can be used to graphically model a system. UML class diagrams are used to represent data and their relationships within the larger UML objectoriented system's modeling language.

2.5.5 NEWER DATA MODELS: OBJECT/RELATIONAL AND XML

Facing the demand to support more complex data representations, the relational model's main vendors evolved the model further and created the extended relational data model (ERDM). The ERDM adds many of the OO model's features within the inherently simpler relational database structure. The ERDM gave birth to a new generation of relational databases supporting OO features such as objects (encapsulated data and methods), extensible data types based on classes, and inheritance. That's why a DBMS based on the ERDM is often described as an object/relational database management system (O/R DBMS).

The use of complex objects received a boost with the Internet revolution. When organizations integrated their business models with the Internet, they realized the potential of the Internet to access, distribute, and exchange critical business information. This resulted in the widespread

adoption of the Internet as a business communication tool. It is in this environment that Extensible Markup Language (XML) emerged as the de facto standard for the efficient and effective exchange of structured, semi structured, and unstructured data. Organizations using XML data soon realized there was a need to manage the large amounts of unstructured data such as word-processing documents, Web pages, e-mails, diagrams, etc., found in most of today's organizations. To address this need, XML databases emerged to manage unstructured data within a native XML format (see Chapter 14, Database Connectivity and Web Technologies, for more information about XML). At the same time, O/R DBMSs added support for XML-based documents within their relational data structure.

2.5.6 THE FUTURE OF DATA MODELS

Today the O/R DBMS is the dominant database for business applications. Its success could be attributed to the model's conceptual simplicity, easy-to-use query language, high transaction performance, high availability, security, scalability, and expandability. In contrast, the OO DBMS is popular in niche markets such as computer-aided drawing/computeraided manufacturing (CAD/CAM), geographic information systems (GIS), telecommunications, and multimedia, which require support for complex objects.

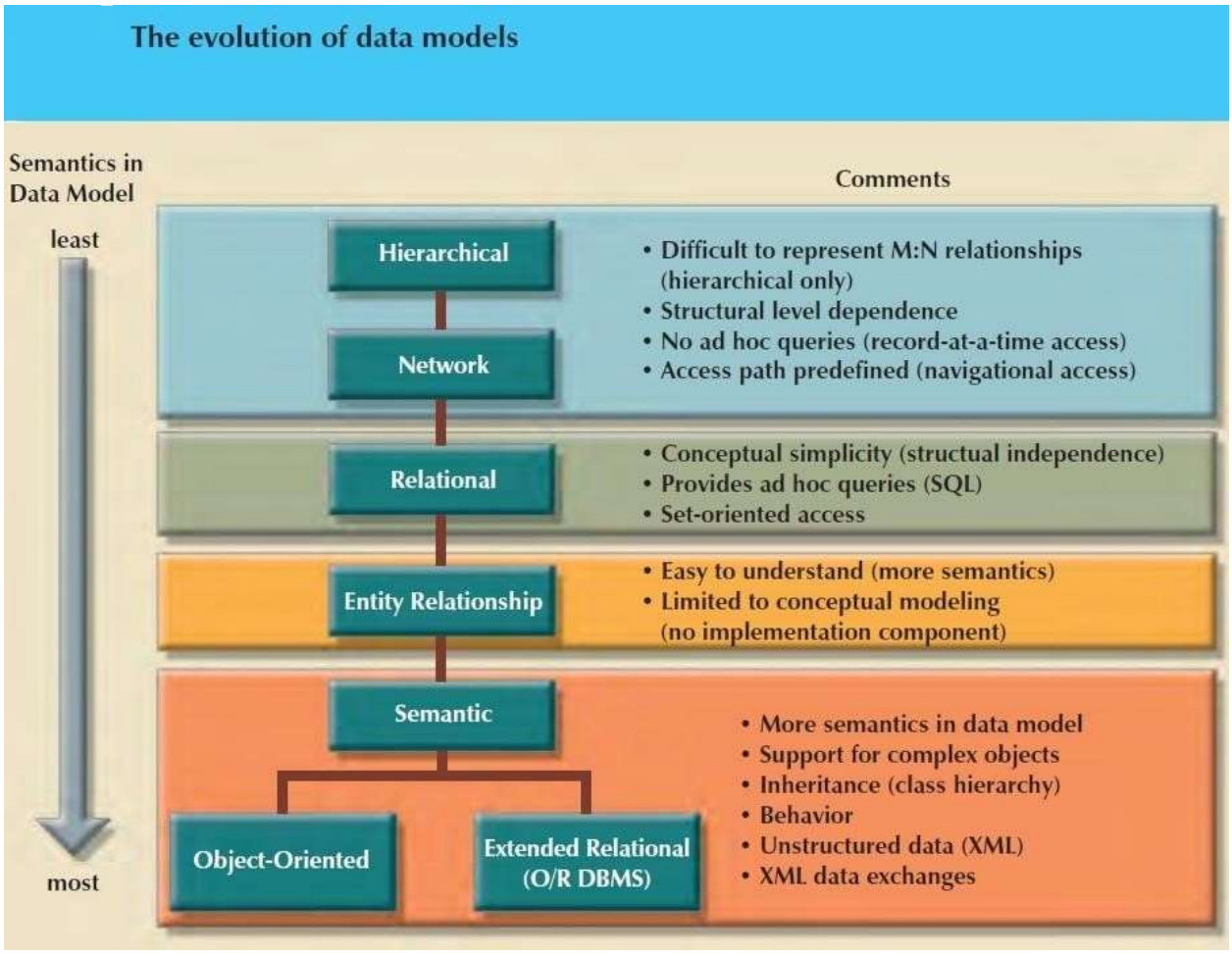
The OO and the relational data models have two totally different approaches. The OO data model was created to address very specific engineering needs, not the wide-ranging needs of general data management tasks. The relational model was created with a focus on better data management based on a sound mathematical foundation. Given these differences, it is not surprising that the growth of the OO market has been slow compared to the rapid growth of the relational data model. One area in which OO concepts have been very influential is systems development and programming languages. Most contemporary programming languages are object-oriented (Java, Ruby, Perl, C#, .NET, to name a few). Also, there is an increasing need to manage an organization's unstructured data. It is difficult to speculate on the future development of database models. Will unstructured data management overcome structured data management? We think that each approach complements and augments the other. O/R databases have proven to efficiently support structured and unstructured data management. Furthermore, history has shown that O/R DBMS are remarkably adaptable in supporting ever-evolving data management needs. Two examples of this evolution are:

- Hybrid DBMSs are emerging that retain the advantages of the relational model and at the same time provide programmers with an object-oriented view of the underlying data. These types of databases preserve the performance characteristics of the relational model and the semantically rich programmatic support of the object-oriented model.
- SQL data services, such as Microsoft SQL Data Services (SDS) on its Azure Services Platform, are becoming a critical component of relational database vendors' Internet service strategies. These "cloud-based" (that is, remotely processed and Internet-based) data services make it possible for companies of any size to store their data in relational databases without incurring expensive hardware, software, and personnel costs, while having access to high-end database features such as failover, backup, high transaction rates, and global data distribution. Companies can use a "pay as you go" system based primarily on their storage and bandwidth utilization and the features used.

2.5.7 DATA MODELS: A SUMMARY

The evolution of DBMSs has always been driven by the search for new ways of modeling increasingly complex real-world data. A summary of the most commonly recognized data models is shown in Figure. In the evolution of data models, there are some common characteristics that data models must have in order to be widely accepted:

- A data model must show some degree of conceptual simplicity without compromising the semantic completeness of the database. It does not make sense to have a data model that is more difficult to conceptualize than the real world.
- A data model must represent the real world as closely as possible. This goal is more easily realized by adding more semantics to the model's data representation. (Semantics concern the dynamic data behavior, while data representation constitutes the static aspect of the real-world scenario.)
- Representation of the real-world transformations (behavior) must be in compliance with the consistency and integrity characteristics of any data model.



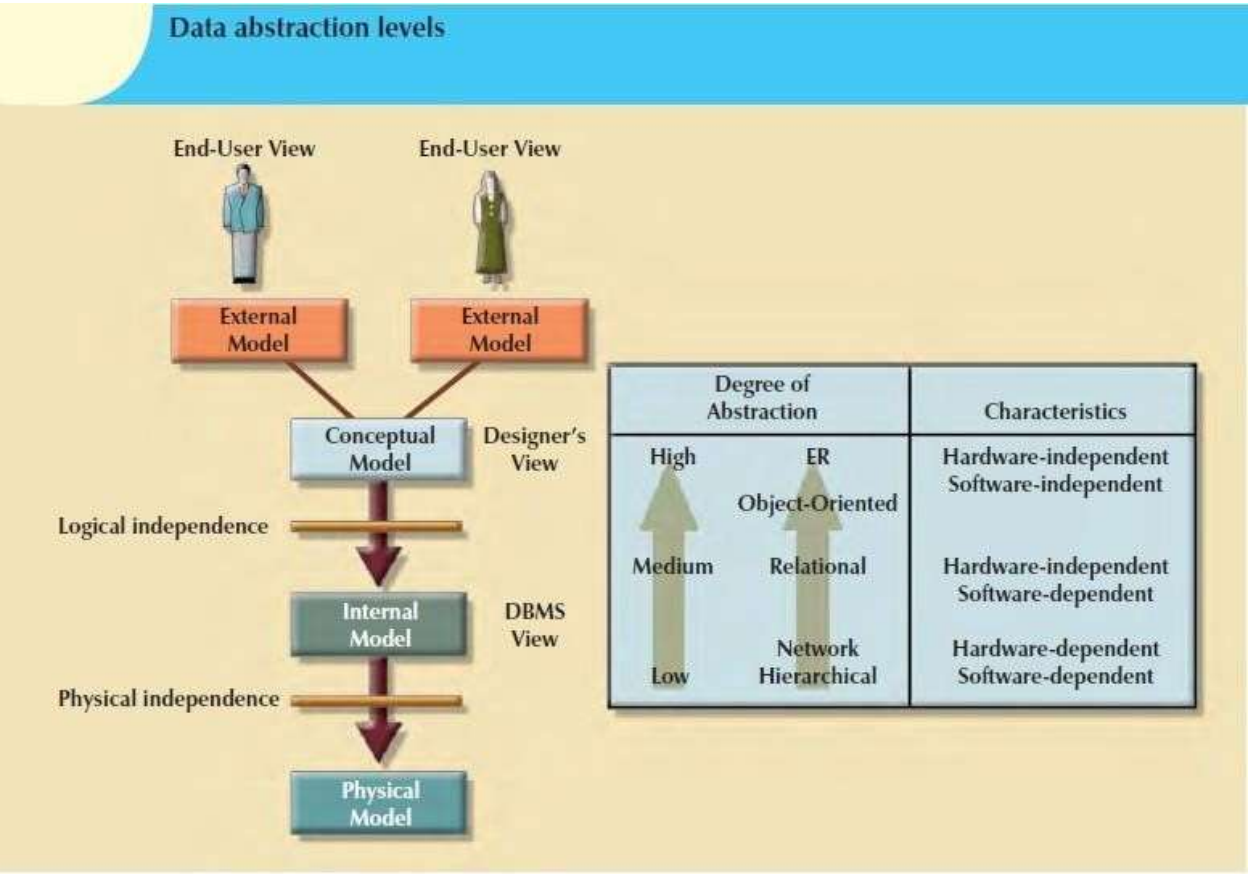
Each new data model addresses the shortcomings of previous models. The network model replaced the hierarchical model because the former made it much easier to represent complex (many-to-many) relationships. In turn, the relational model offers several advantages over the hierarchical and network models through its simpler data representation, superior data independence, and easy-to-use query language; these features made it the preferred data model for business applications. The OO data model introduced support for complex data within a rich semantic framework. The

ERDM added many of the OO features to the relational model and allowed it to maintain its strong market share within the business environment. And in recent years, successful data models have facilitated the development of database products that incorporate unstructured data as well as provide support for easy data exchanges via XML. It is important to note that not all data models are created equal; some data models are better suited than others for some tasks. For example, conceptual models are better suited for high-level data modeling, while implementation models are better for managing stored data for implementation purposes. The entity relationship model is an example of a conceptual model, while the hierarchical and network models are examples of implementation models. At the same time, some models, such as the relational model and the OODM, could be used as both conceptual and implementation models. Table summarizes the advantages and disadvantages of the various database models.

Advantages and Disadvantages of Various Database Models

DATA MODEL	DATA INDEPENDENCE	STRUCTURAL INDEPENDENCE	ADVANTAGES	DISADVANTAGES
Hierarchical	Yes	No	1. It promotes data sharing. 2. Parent/Child relationship promotes conceptual simplicity. 3. Database security is provided and enforced by DBMS. 4. Parent/Child relationship promotes data integrity. 5. It is efficient with 1:M relationships.	1. Complex implementation requires knowledge of physical data storage characteristics. 2. Navigational system yields complex application development, management, and use; requires knowledge of hierarchical path. 3. Changes in structure require changes in all application programs. 4. There are implementation limitations (no multiparent or M:N relationships). 5. There is no data definition or data manipulation language in the DBMS. 6. There is a lack of standards.
Network	Yes	No	1. Conceptual simplicity is at least equal to that of the hierarchical model. 2. It handles more relationship types, such as M:N and multiparent. 3. Data access is more flexible than in hierarchical and file system models. 4. Data Owner/Member relationship promotes data integrity. 5. There is conformance to standards. 6. It includes data definition language (DDL) and data manipulation language (DML) in DBMS.	1. System complexity limits efficiency—still a navigational system. 2. Navigational system yields complex implementation, application development, and management. 3. Structural changes require changes in all application programs.
Relational	Yes	Yes	1. Structural independence is promoted by the use of independent tables. Changes in a table's structure do not affect data access or application programs. 2. Tabular view substantially improves conceptual simplicity, thereby promoting easier database design, implementation, management, and use. 3. Ad hoc query capability is based on SQL. 4. Powerful RDBMS isolates the end user from physical-level details and improves implementation and management simplicity.	1. The RDBMS requires substantial hardware and system software overhead. 2. Conceptual simplicity gives relatively untrained people the tools to use a good system poorly, and if unchecked, it may produce the same data anomalies found in file systems. 3. It may promote "islands of information" problems as individuals and departments can easily develop their own applications.
Entity relationship	Yes	Yes	1. Visual modeling yields exceptional conceptual simplicity. 2. Visual representation makes it an effective communication tool. 3. It is integrated with dominant relational model.	1. There is limited constraint representation. 2. There is limited relationship representation. 3. There is no data manipulation language. 4. Loss of information content occurs when attributes are removed from entities to avoid crowded displays. (This limitation has been addressed in subsequent graphical versions.)
Object-oriented	Yes	Yes	1. Semantic content is added. 2. Visual representation includes semantic content. 3. Inheritance promotes data integrity.	1. Slow development of standards caused vendors to supply their own enhancements, thus eliminating a widely accepted standard. 2. It is a complex navigational system. 3. There is a steep learning curve. 4. High system overhead slows transactions.

subset of the overall data in the organization. Therefore, end users working within those business units view their data subsets as separate from or external to other units within the organization. Because data are being modeled, ER diagrams will be used to represent the external views. A specific representation of an external view is known as an external schema. To illustrate the external model's view, examine the data environment of Tiny College. Figure 2.7 presents the external schemas for two Tiny College business units: student registration and class scheduling. Each external schema includes the appropriate entities, relationships, processes, and constraints imposed by the business unit. Also note that although the application views are isolated from each other, each view shares a common entity with the other view. For example, the registration and scheduling external schemas share the entities CLASS and COURSE.



2.9.2 THE CONCEPTUAL MODEL

Having identified the external views, a conceptual model is used, graphically represented by an ERD, to integrate all external views into a single view. The conceptual model represents a global view of the entire database as viewed by the entire organization. That is, the conceptual model integrates all external views (entities, relationships, constraints, and processes) into a single global view of the data in the enterprise. Also known as a conceptual schema, it is the basis for the identification and high-level description of the main data objects (avoiding any database model–specific details). The most widely used conceptual model is the ER model. Remember that the ER model is illustrated with the help of the ERD, which is, in effect, the basic database blueprint. The ERD is used to graphically represent the conceptual schema. The conceptual model yields some very important advantages. First, it provides a relatively easily understood bird's-eye (macro level) view of the data environment. For example, you can get a summary of
Tiny College's data environment by examining the conceptual model presented in Figure 2.7. Second, the conceptual model is independent of both software and hardware. Software independence means that the model does not depend on the DBMS software used to implement the model. Hardware independence means that the model does not depend on the hardware used in the implementation of the model. Therefore, changes in either the hardware or the DBMS software will have no effect on the database design at the conceptual level. Generally, the term logical design is used to refer to the task of creating a conceptual data model that could be implemented in any DBMS.

2.9.3 THE INTERNAL MODEL

Once a specific DBMS has been selected, the internal model maps the conceptual model to the DBMS. The internal model is the representation of the database as "seen" by the DBMS. In other words, the internal model requires the designer to match the conceptual model's characteristics and constraints to those of the selected implementation model. An internal schema depicts a specific representation of an internal model, using the database constructs supported by the chosen database. Because this book focuses on the relational model, a relational database was chosen to implement the internal model. Therefore, the internal schema should map the conceptual model to the relational model constructs. In particular, the entities in the conceptual model are mapped to tables in the relational model. Likewise, because a relational database has been selected, the internal schema is expressed using SQL, the standard language for relational databases. In the case of the conceptual model for Tiny College depicted in Figure 2.8, the internal model was implemented by creating the tables PROFESSOR, COURSE, CLASS, STUDENT, ENROLL, and ROOM.

The development of a detailed internal model is especially important to database designers who work with hierarchical or network models because those models require very precise specification of data storage location and data access paths. In contrast, the relational model requires less detail in its internal model because most RDBMSs handle data access path definition transparently; that is, the designer need not be aware of the data access path details.

Nevertheless, even relational database software usually requires data storage location specification, especially in a mainframe environment. For example, DB2 requires that you specify the data storage group, the location of the database within the storage group, and the location of the tables within the database. Because the internal model depends on specific database software, it is said to be software-dependent. Therefore, a change in the DBMS software requires that the internal model be changed to fit the characteristics and requirements of the implementation database model. When you can change the internal model without affecting the conceptual model, you have logical independence. However, the internal model is still hardware-independent because it is unaffected by the choice of the computer on which the software is installed. Therefore, a change in storage devices or even a change in operating systems will not affect the internal model.

2.9.4 THE PHYSICAL MODEL

The physical model operates at the lowest level of abstraction, describing the way data are saved on storage media such as disks or tapes. The physical model requires the definition of both the physical storage devices and the (physical) access methods required to reach the data within those storage devices, making it both software- and hardwaredependent. The storage structures used are dependent on the software (the DBMS and the operating system) and on the type of storage devices that the computer can handle. The precision required in the physical model's definition demands that database designers who work at this level have a detailed knowledge of the hardware and software used to implement the database design. Early data models forced the database designer to take the details of the physical model's data storage requirements into account. However, the now dominant relational model is aimed largely at the logical rather than the physical level; therefore, it does not require the physical-level details common to its predecessors. Although the relational model does not require the designer to be concerned about the data's physical storage characteristics, the implementation of a relational model may require physical-level fine-tuning for increased performance. Fine-tuning is especially important when very large databases are installed

in a mainframe environment. Yet even such performance finetuning at the physical level does not require knowledge of physical data storage characteristics.

As noted earlier, the physical model is dependent on the DBMS, methods of accessing files, and types of hardware storage devices supported by the operating system. When you can change the physical model without affecting the internal model, you have physical independence. Therefore, a change in storage devices or methods and even a change in operating system will not affect the internal model.

SUMMARY

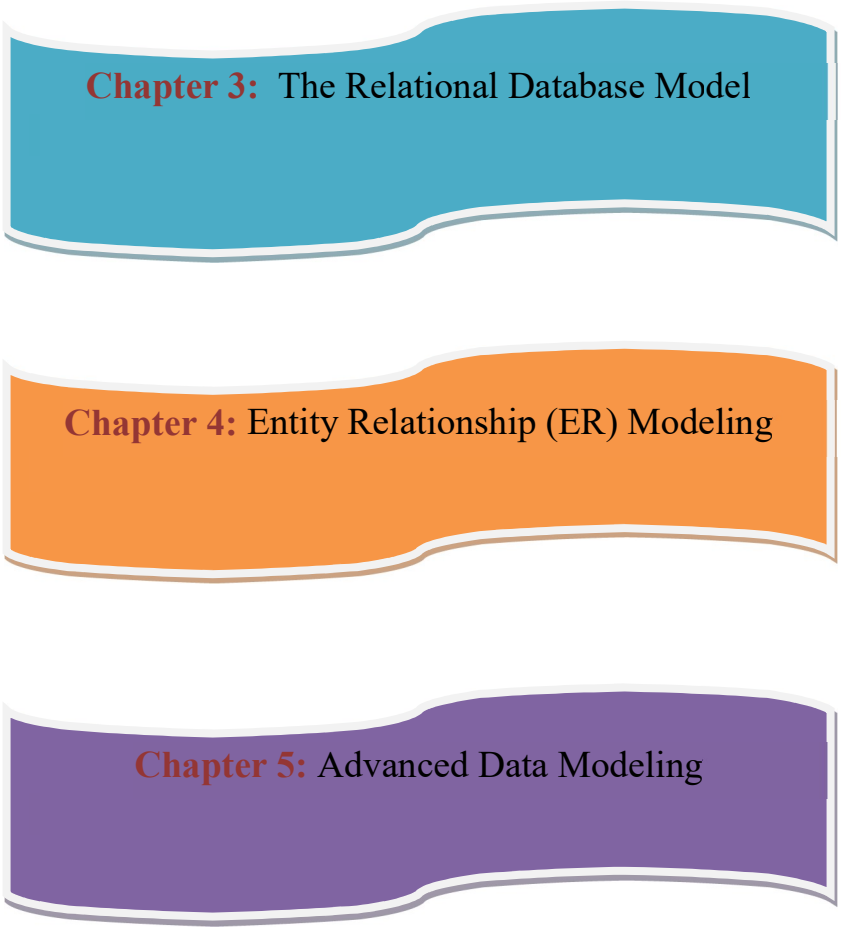
A data model is an abstraction of a complex real-world data environment. Database designers use data models to communicate with applications programmers and end users. The basic datamodeling components are entities, attributes, relationships, and constraints. Business rules are used to identify and define the basic modeling components within a specific real-world environment.

- The hierarchical and network data models were early data models that are no longer used, but some of the concepts are found in current data models. The hierarchical model depicts a set of one-to-many (1:M) relationships between a parent and its children segments. The network model uses sets to represent 1:M relationships between record types.
- The relational model is the current database implementation standard. In the relational model, the end user perceives the data as being stored in tables. Tables are related to each other by means of common values in common attributes. The entity relationship (ER) model is a popular graphical tool for data modeling that complements the relational model. The ER model allows database designers to visually present different views of the data—as seen by database designers, programmers, and end users—and to integrate the data into a common framework.
- The object-oriented data model (OODM) uses objects as the basic modeling structure. An object resembles an entity in that it includes the facts that define it. But unlike an entity, the object also includes information about relationships between the facts, as well as relationships with other objects, thus giving its data more meaning.
- The relational model has adopted many object-oriented (OO) extensions to become the extended relational data model (ERDM). Object/relational database management systems (O/R DBMS) were developed to implement the ERDM. At this point, the OODM is largely used in specialized engineering and scientific applications, while the ERDM is primarily geared to business applications. Although the most likely future scenario is an increasing merger of OODM and ERDM technologies, both are overshadowed by the need to develop Internet access strategies for databases. Usually OO data models are depicted using Unified Modeling Language (UML) class diagrams.
- Data-modeling requirements are a function of different data views (global vs. local) and the level of data abstraction. The American National Standards Institute Standards Planning and Requirements Committee (ANSI/SPARC) describes three levels of data abstraction: external, conceptual, and internal. There is also a fourth level of data abstraction, the physical level. This lowest level of data abstraction is concerned exclusively with physical storage methods.

REVIEW QUESTIONS

1. Discuss the importance of data modeling.
2. What is a business rule, and what is its purpose in data modeling?
3. How do you translate business rules into data model components?
4. What languages emerged to standardize the basic network data model, and why was such standardization important to users and designers?
5. Describe the basic features of the relational data model and discuss their importance to the end user and the designer.
9. Explain how the entity relationship (ER) model helped produce a more structured relational database design environment.
9. Use the scenario described by "A customer can make many payments, but each payment is made by only one customer" as the basis for an entity relationship diagram (ERD) representation.
7. Why is an object said to have greater semantic content than an entity?
8. What is the difference between an object and a class in the object-oriented data model (OODM)?
10. How would you model Question 7 with an OODM? (Use Figure 2.4 as your guide.)
11. What is an ERDM, and what role does it play in the modern (production) database environment?
12. In terms of data and structural independence, compare file system data management with the five data models discussed in this chapter.
13. What is a relationship, and what three types of relationships exist?
14. Give an example of each of the three types of relationships.
15. What is a table, and what role does it play in the relational model?
19. What is a relational diagram? Give an example.
19. What is logical independence?
17. What is physical independence?
18. What is connectivity? (Use a Crow's Foot ERD to illustrate connectivity.)

Module 2:
Entity-Relationship and Relational Database Model



Chapter 3: The Relational Database Model

3.1 A Logical View of Data

Database Systems, you learned that a database stores and manages both data and metadata. You also learned that the DBMS manages and controls access to the data and the database structure. Such an arrangement— placing the DBMS between the application and the database—eliminates most of the file system's inherent limitations. The result of such flexibility, however, is a far more complex physical structure. In fact, the database structures required by both the hierarchical and network database models often become complicated enough to diminish efficient database design. The relational data model changed all of that by allowing the designer to focus on the logical representation of the data and its relationships, rather than on the physical storage details. To use an automotive analogy, the relational database uses an automatic transmission to relieve you of the need to manipulate clutch pedals and gear shifts. In short, the relational model enables you to view data *logically* rather than *physically*. The practical significance of taking the logical view is that it serves as a reminder of the simple file concept of data storage. Although the use of a table, quite unlike that of a file, has the advantages of structural and data independence, a able does resemble

a file from a conceptual point of view. Because you can think of related records as being stored in independent tables, the relational database model is much easier to understand than the hierarchical and network models. Logical simplicity tends to yield simple and effective database design methodologies. Because the table plays such a prominent role in the relational model, it deserves a closer look. Therefore, our discussion begins with an exploration of the details of table structure and contents.

3.1.1 Tables and Their Characteristics

The logical view of the relational database is facilitated by the creation of data relationships based on a logical construct known as a relation. Because a relation is a mathematical construct, end users find it much easier to think of a relation as a table. A table is perceived as a twodimensional structure composed of rows and columns. A table is also called a *relation* because the relational model's creator, E. F. Codd, used the term *relation* as a synonym for table. You can think of a table as a *persistent* representation of a logical relation, that is, a relation whose contents can be permanently saved for future use. As far as the table's user is concerned, *a table contains a group of related entity occurrences*, that is, an entity set. For example, a STUDENT table contains a collection of entity occurrences, each representing a student. For that reason, the terms *entity set* and *table* are often used interchangeably. You will discover that the table view of data makes it easy to spot and define entity relationships, thereby greatly simplifying the task of database design. The characteristics of a relational table are summarized in below Table:

Characteristics of a Relational Table

1	A table is perceived as a two-dimensional structure composed of rows and columns.
2	Each table row (**tuple**) represents a single entity occurrence within the entity set.
3	Each table column represents an attribute, and each column has a distinct name.
4	Each row/column intersection represents a single data value.
5	All values in a column must conform to the same data format.
6	Each column has a specific range of values known as the **attribute domain**.
7	The order of the rows and columns is immaterial to the DBMS.
8	Each table must have an attribute or a combination of attributes that uniquely identifies each row.

The table shown in Figure below illustrates the characteristics listed in Table above.

STUDENT table attribute values

Table name: STUDENT — Database name: Ch03_TinyCollege

STU_NUM	STU_LNAME	STU_FNAME	STU_INIT	STU_DOB	STU_HRS	STU_CLASS	STU_GPA	STU_TRANSFER	DEPT_CODE	STU_PHONE	PROF_NUM
321452	Bowser	William	C	12-Feb-1975	42	So	2.84	No	BIOL	2134	205
324257	Smithson	Anne	K	15-Nov-1981	81	Jr	3.27	Yes	CIS	2256	222
324258	Brewer	Juliette		23-Aug-1969	36	So	2.26	Yes	ACCT	2256	228
324269	Oblonski	Walter	H	16-Sep-1976	66	Jr	3.09	No	CIS	2114	222
324273	Smith	John	D	30-Dec-1958	102	Sr	2.11	Yes	ENGL	2231	199
324274	Katinga	Raphael	P	21-Oct-1979	114	Sr	3.15	No	ACCT	2267	228
324291	Robertson	Gerald	T	08-Apr-1973	120	Sr	3.87	No	EDU	2267	311
324299	Smith	John	B	30-Nov-1986	15	Fr	2.92	No	ACCT	2315	230

STU_NUM = Student number
STU_LNAME = Student last name
STU_FNAME = Student first name
STU_INIT = Student middle initial
STU_DOB = Student date of birth
STU_HRS = Credit hours earned
STU_CLASS = Student classification
STU_GPA = Grade point average
STU_TRANSFER = Student transferred from another institution
DEPT_CODE = Department code
STU_PHONE = 4-digit campus phone extension
PROF_NUM = Number of the professor who is the student's advisor

3.2 Keys

In the relational model, keys are important because they are used to ensure that each row in a table is uniquely identifiable. They are also used to establish relationships among tables and to ensure the integrity of the data. Therefore, a proper understanding of the concept and use of keys in the relational model is very important. A **key** consists of one or more attributes that determine other attributes. For example, an invoice number identifies all of the invoice attributes, such as the invoice date and the customer name. One type of key, the primary key, has already been introduced. Given the structure of the STUDENT table shown in Figure 3.1, defining and describing the primary key seem simple enough. However, because the primary key plays such an important role in the relational environment, you will examine the primary key's properties more carefully. In this section, you also will become acquainted with superkeys, candidate keys, and secondary keys. The key's role is based on a concept known as **determination**. In the context of a database table, the statement "A determines B" indicates that if you know the value of attribute A, you can look up (determine) the value of attribute B. For example, knowing the
STU_NUM in the STUDENT table (see Figure 3.1) means that you are able to look up (determine) that student's last name, grade point average, phone number, and so on. The shorthand notation for "A determines B" is A → B. If A determines B, C, and D, you write A → B, C, D. Therefore, using the attributes of the STUDENT table in Figure 3.1, you can represent the statement "STU_NUM determines STU_LNAME" by writing:
STU_NUM → STU_LNAME
In fact, the STU_NUM value in the STUDENT table determines all of the student's attribute values. For example, you can write: STU_NUM → STU_LNAME, STU_FNAME, STU_INIT and STU_NUM → STU_LNAME, STU_FNAME, STU_INIT, STU_DOB, STU_TRANSFER
In contrast, STU_NUM is not determined by STU_LNAME because it is quite possible for several students to have the last name Smith.
The principle of determination is very important because it is used in the definition of a central relational database concept known as functional dependence. The term **functional dependence**

can be defined most easily this way: the attribute B is functionally dependent on A if A determines B. More precisely:

The attribute B is functionally dependent on the attribute A if each value in column A determines *one and only one* value in column B.

Using the contents of the STUDENT table in Figure, it is appropriate to say that STU_PHONE is functionally dependent on STU_NUM. For example, the STU_NUM value 321452 determines the STU_PHONE value 2134. On the other hand, STU_NUM is not functionally dependent on STU_PHONE because the STU_PHONE value 2267 is associated with two STU_NUM values: 324274 and 324291. (This could happen when roommates share a single land line phone number.) Similarly, the STU_NUM value 324273 determines the STU_LNAME value Smith. But the STU_NUM value is not functionally dependent on STU_LNAME because more than one student may have the last name Smith. The functional dependence definition can be generalized to cover the case in which the determining attribute values occur more than once in a table. Functional dependence can then be defined this way:1

Attribute A determines attribute B (that is, B is functionally dependent on A) if all of the rows in the table that agree in value for attribute A also agree in value for attribute B. Be careful when defining the dependency's direction. For example, Gigantic State University determines its student classification based on hours completed; these are shown in Table below.

Student Classification

HOURS COMPLETED	CLASSIFICATION
Less than 30	Fr
30–59	So
60–89	Jr
90 or more	Sr

Therefore, you can write: STU_HRS → STU_CLASS But the specific number of hours is not dependent on the classification. It is quite possible to find a junior with 62 completed hours or one with 84 completed hours. In other words, the classification (STU_CLASS) does not determine one and only one value for completed hours (STU_HRS). Keep in mind that it might take more than a single attribute to define functional dependence; that is, a key may be composed of more than one attribute. Such a multi attribute key is known as a **composite key**. Any attribute that is part of a key is known as a **key attribute**. For instance, in the STUDENT table, the student's last name would not be sufficient to serve as a key. On the other hand, the combination of last name, first name, initial, and phone is very likely to produce unique matches for the remaining attributes. For example, you can write: STU_LNAME, STU_FNAME, STU_INIT, STU_PHONE → STU_HRS, STU_CLASS or STU_LNAME, STU_FNAME, STU_INIT, STU_PHONE → STU_HRS, STU_CLASS, STU_GPA or STU_LNAME, STU_FNAME,

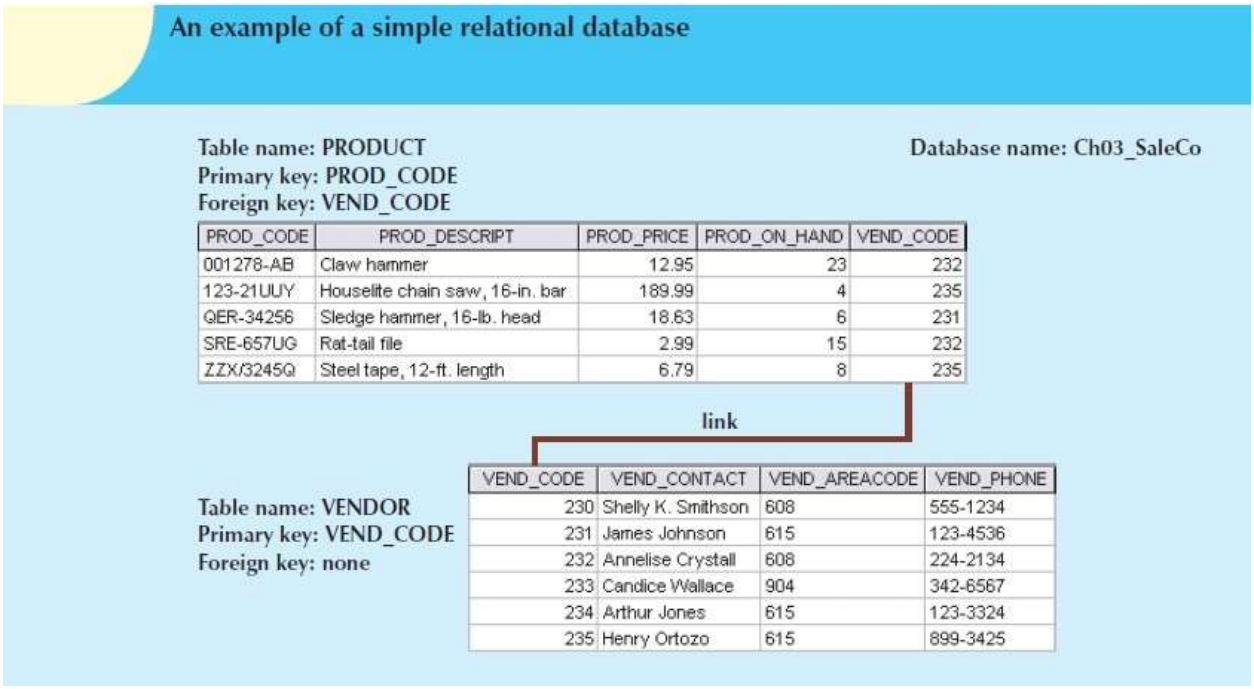
STU_INIT, STU_PHONE → STU_HRS, STU_CLASS, STU_GPA, STU_DOB

Given the possible existence of a composite key, the notion of functional dependence can be further refined by specifying **full functional dependence**: **If the attribute (B) is functionally dependent on a composite key (A) but not on any subset of that composite key, the attribute (B) is fully functionally dependent on (A).** Within the broad key classification, several specialized keys can be defined. For example, a **superkey** is any key that uniquely identifies each row. In short, the superkey functionally determines all of a row's attributes. In the STUDENT table, the superkey could be any of the following:

STU_NUM
STU_NUM, STU_LNAME
STU_NUM, STU_LNAME, STU_INIT
In fact, STU_NUM, with or without additional attributes, can be a superkey even when the additional attributes are redundant. A **candidate key** can be described as a superkey without unnecessary attributes, that is, a minimal superkey. Using this distinction, note that the composite key STU_NUM, STU_LNAME is a superkey, but it is not a candidate key because STU_NUM by itself is a candidate key! The combination STU_LNAME, STU_FNAME, STU_INIT, STU_PHONE might also be a candidate key, as long as you discount the possibility that two students share the same last name, first name, initial, and phone number.
If the student's Social Security number had been included as one of the attributes in the STUDENT table in Figure 3.1—perhaps named STU_SSN—both it and STU_NUM would have been candidate keys because either one would uniquely identify each student. In that case, the selection of STU_NUM as the primary key would be driven by the designer's choice or by enduser requirements. In short, the primary key is the candidate key chosen to be the unique row identifier. Note, incidentally, that a primary key is a superkey as well as a candidate key. Within a table, each primary key value must be unique to ensure that each row is uniquely identified by the primary key. In that case, the table is said to exhibit **entity integrity**. To maintain entity integrity, a **null** (that is, no data entry at all) is not permitted in the primary key.

Nulls can *never* be part of a primary key, and they should be avoided—to the greatest extent possible—in other attributes, too. There are rare cases in which nulls cannot be reasonably avoided when you are working with nonkey attributes. For example, one of an EMPLOYEE table's attributes is likely to be the EMP_INITIAL. However, some employees do not have a middle initial. Therefore, some of the EMP_INITIAL values may be null. You will also discover later in this section that there may be situations in which a null exists because of the nature of the relationship between two entities. In any case, even if nulls cannot always be avoided, they must be used sparingly. In fact, the existence of nulls in a table is often an indication of poor database design. Nulls, if used improperly, can create problems because they have many different meanings. For example, a null can represent:
_ An unknown attribute value.
_ A known, but missing, attribute value.
_ A "not applicable" condition.
Depending on the sophistication of the application development software, nulls can create problems when functions such as COUNT, AVERAGE, and SUM are used. In addition, nulls can create logical problems when relational tables are linked. *Controlled redundancy* makes the relational database work. Tables within the database share common attributes that enable the tables to be linked together. For example, note that the PRODUCT and VENDOR tables in Figure 3.2 share a common attribute named VEND_CODE. And note that the PRODUCT table's VEND_CODE value 232 occurs more than once, as does the VEND_CODE value 235. Because the PRODUCT table is related to the VENDOR table through these VEND_CODE values, the multiple occurrence of the values is *required* to make the 1:M relationship between VENDOR and PRODUCT work. Each VEND_CODE value in the VENDOR table is unique—the
VENDOR is the "1" side in the VENDOR-PRODUCT relationship. But any given VEND_CODE value from the VENDOR table may occur more than once in the PRODUCT table, thus providing evidence that PRODUCT is the "M" side of the VENDOR-PRODUCT relationship. In database terms, the multiple occurrences of the VEND_CODE values in the PRODUCT table are not

redundant because they are *required* to make the relationship work. You should recall from that data redundancy exists only when there is *unnecessary* duplication of attribute values.



As you examine Figure 3.2, note that the VEND_CODE value in one table can be used to point to the corresponding
value in the other table. For example, the VEND_CODE value 235 in the PRODUCT table points to vendor Henry Ortozo in the VENDOR table. Consequently, you discover that the product "Houselite chain saw, 16-in. bar" is delivered by Henry Ortozo and that he can be contacted by calling 615-899-3425. The same connection can be made for the product "Steel tape, 12-ft. length" in the PRODUCT table. Remember the naming convention—the prefix
PROD was used in Figure 3.2 to indicate that the attributes "belong" to the PRODUCT table. Therefore, the prefix VEND in the PRODUCT table's VEND_CODE indicates that VEND_CODE points to some other table in the database. In this case, the VEND prefix is used to point to the VENDOR table in the database.

A relational database can also be represented by a relational schema. A **relational schema** is a textual representation of the database tables where each table is listed by its name followed by the list of its attributes in parentheses. The primary key attribute(s) is (are) underlined. You will see such schemas in Chapter 6, Normalization of Database Tables. For example, the relational schema for Figure 3.2 would be shown as: VENDOR (**VEND_CODE**, VEND_CONTACT, VEND_AREACODE, VEND_PHONE) PRODUCT (**PROD_CODE**, PROD_DESCRIPT, PROD_PRICE, PROD_ON_HAND, VEND_CODE) The link between the PRODUCT and VENDOR tables in Figure 3.2 can also be represented by the relational diagram shown in Figure 3.3. In this case, the link is indicated by the line that connects the VENDOR and PRODUCT tables. Note that the link in Figure 3.3 is the equivalent of the relationship line in an ERD. This link is created when two tables share an attribute with common values. More specifically, the primary key of one table (VENDOR) appears as the *foreign key* in a related table (PRODUCT). A **foreign key** (**FK**) is an attribute whose values match the primary key values in the related table. For example, in Figure 3.2, the VEND_CODE is the primary key in the VENDOR table, Finally, a **secondary key** is defined as a key that is used strictly for data retrieval purposes. Suppose customer data are stored in a CUSTOMER table in which the customer number is the primary key. Do you suppose that most customers will remember their numbers? Data retrieval for a customer can be

facilitated when the customer's last name and phone number are used. In that case, the primary key is the customer number; the secondary key is the combination of the customer's last name and phone number. Keep in mind that a secondary key does not necessarily yield a unique outcome. For example, a customer's last name and home telephone number could easily yield several matches where one family lives together and shares a phone line. A less efficient secondary key would be the combination of the last name and zip code; this could yield dozens of matches, which could then be combed for a specific match. A secondary key's effectiveness in narrowing down a search depends on how restrictive that secondary key is. For instance, although the secondary key CUS_CITY is legitimate from a database point of view, the attribute values "New York" or "Sydney" are not likely to produce a usable return unless you want to examine millions of possible matches. (Of course, CUS_CITY is a better secondary key than CUS_COUNTRY.) Table below summarizes the various relational database table keys.

TABLE **Relational Database Keys**

KEY TYPE	DEFINITION
Superkey	An attribute (or combination of attributes) that uniquely identifies each row in a table.
Candidate key	A minimal (irreducible) superkey. A superkey that does not contain a subset of attributes that is itself a superkey.
Primary key	A candidate key selected to uniquely identify all other attribute values in any given row. Cannot contain null entries.
Secondary key	An attribute (or combination of attributes) used strictly for data retrieval purposes.
Foreign key	An attribute (or combination of attributes) in one table whose values must either match the primary key in another table or be null.

3.3 Integrity Rules

Relational database integrity rules are very important to good database design. Many (but by no means all) RDBMSs enforce integrity rules automatically. However, it is much safer to make sure that your application design conforms to the entity and referential integrity rules mentioned in this chapter. Those rules are summarized in Table below:

TABLE **Integrity Rules**

ENTITY INTEGRITY	DESCRIPTION
Requirement	All primary key entries are unique, and no part of a primary key may be null.
Purpose	Each row will have a unique identity, and foreign key values can properly reference primary key values.
Example	No invoice can have a duplicate number, nor can it be null. In short, all invoices are uniquely identified by their invoice number.
REFERENCE INTEGRITY	**DESCRIPTION**
Requirement	A foreign key may have either a null entry, as long as it is not a part of its table's primary key, or an entry that matches the primary key value in a table to which it is related. (Every non-null foreign key value *must* reference an *existing* primary key value.)
Purpose	It is possible for an attribute NOT to have a corresponding value, but it will be impossible to have an invalid entry. The enforcement of the referential integrity rule makes it impossible to delete a row in one table whose primary key has mandatory matching foreign key values in another table.
Example	A customer might not yet have an assigned sales representative (number), but it will be impossible to have an invalid sales representative (number).

3.4 Relational Set Operators

The data in relational tables are of limited value unless the data can be manipulated to generate useful information. This section describes the basic data manipulation capabilities of the relational model. **Relational algebra** defines the theoretical way of manipulating table contents using the eight relational operators: SELECT, PROJECT, JOIN, INTERSECT, UNION, DIFFERENCE, PRODUCT, and DIVIDE. In Chapter 7, Introduction to Structured Query Language (SQL), and Chapter 8, Advanced SQL, you will learn how SQL commands can be used to accomplish relational algebra operations. The relational operators have the property of **closure**; that is, the use of relational algebra operators on existing relations (tables) produces new relations. There is no need to examine the mathematical definitions, properties, and characteristics of those relational algebra operators. However, their *use* can easily be illustrated as follows:

REF: Database Systems: Design, Implementation, and Management, Ninth Edition Carlos Coronel, Steven Morris, and Peter Rob (Page 69)

3.5 The Data Dictionary and the System Catalog

The **data dictionary** provides a detailed description of all tables found within the user/designercreated database. Thus, the data dictionary contains at least all of the attribute names and characteristics for each table in the system. In short, the data dictionary contains metadata—data about data. Using the small database presented in Figure 3.4, you might picture its data dictionary as shown in Table 3.6. The data dictionary is sometimes described as "the database designer's database" because it records the design decisions about tables and their structures. Like the data dictionary, the **system catalog** contains metadata. The system catalog can be described as a detailed system data dictionary that describes all objects within the database, including data about table names, the table's creator and creation date, the number of columns in each table, the data type corresponding to each column, index filenames, index creators,

authorized users, and access privileges. Because the system catalog contains all required data dictionary information, the terms *system catalog* and *data dictionary* are often used interchangeably. In fact, current relational database software generally provides only a system catalog, from which the designer's data dictionary information may be derived. The system catalog is actually a system-created database whose tables store the user/designer-created database characteristics and contents. Therefore, the system catalog tables can be queried just like any user/designer-created table. In effect, the system catalog automatically produces database documentation. As new tables are added to the database, that documentation also allows the RDBMS to check for and eliminate homonyms and synonyms. In general terms, **homonyms** are similar-sounding words with different meanings, such as *boar* and *bore*, or identically spelled words with different meanings, such as *fair* (meaning "just") and *fair* (meaning "festival"). In a database context, the word *homonym* indicates the use of the same attribute name to label different attributes. For example, you might use C_NAME to label a customer name attribute in a CUSTOMER table and also use C_NAME to label a consultant name attribute in a CONSULTANT table. To lessen confusion, you should avoid database homonyms; the data dictionary is very useful in this regard. In a database context, a **synonym** is the opposite of a homonym and indicates the use of different names to describe the same attribute. For example, *car* and *auto* refer to the same object. Synonyms must be avoided.

REF: Database Systems: Design, Implementation, and Management, Ninth Edition Carlos Coronel, Steven Morris, and Peter Rob (Page 74).

3.6 Relationships within the Relational Database

You already know that relationships are classified as one-to-one (1:1), one-to-many (1:M), and many-to-many (M:N or M:M). This section explores those relationships further to help you apply them properly when you start developing database designs, focusing on the following points: _

The 1:M relationship is the relational modeling ideal. Therefore, this relationship type should be the norm in any relational database design.

_ The 1:1 relationship should be rare in any relational database design.

_ M:N relationships cannot be implemented as such in the relational model. Later in this section, you will see how any M:N relationship can be changed into two 1:M relationships.

3.6.1 The 1:M Relationship

3.6.2 The 1:1 Relationship

3.6.3 The M:N Relationship

REF: Database Systems: Design, Implementation, and Management, Ninth Edition Carlos Coronel, Steven Morris, and Peter Rob (Page 76-83)

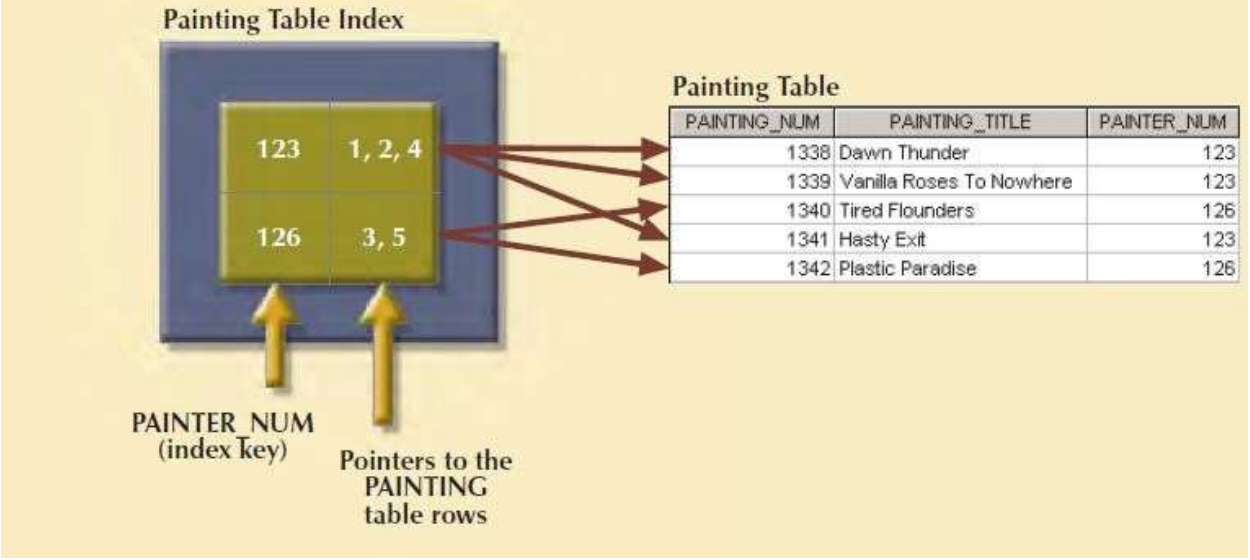
3.7 Data Redundancy Revisited

You learned that data redundancy leads to data anomalies. Those anomalies can destroy the effectiveness of the database. You also learned that the relational database makes it possible to control data redundancies by using common attributes that are shared by tables, called foreign keys. The proper use of foreign keys is crucial to controlling data redundancy. Although the use of foreign keys does not totally eliminate data redundancies, because the foreign key values can be repeated many times, the proper use of foreign keys minimizes data redundancies, thus minimizing the chance that destructive data anomalies will develop.

3.8 Indexes

Suppose you want to locate a particular book in a library. Does it make sense to look through every book in the library
until you find the one you want? Of course not; you use the library's catalog, which is indexed by title, topic, and
author. The index (in either a manual or a computer system) points you to the book's location, thereby making retrieval of the book a quick and simple matter. An index is an orderly arrangement used to logically access rows in a table. Or suppose you want to find a topic, such as "ER model," in this book. Does it make sense to read through every page
until you stumble across the topic? Of course not; it is much simpler to go to the book's index, look up the phrase ER model, and read the page references that point you to the appropriate page(s). In each case, an index is used to locate a needed item quickly. Indexes in the relational database environment work like the indexes described in the preceding paragraphs. From a conceptual point of view, an index is composed of an index key and a set of pointers. The index key is, in effect, the index's reference point. More formally, an index is an ordered arrangement of keys and pointers. Each key points to the location of the data identified by the key. For example, suppose you want to look up all of the paintings created by a given painter in the Ch03_Museum database in Figure 3.19. Without an index, you must read each row in the PAINTING table and see if the PAINTER_NUM matches the requested painter. However, if you index the PAINTER table and use the index key PAINTER_NUM, you merely need to look up the appropriate PAINTER_NUM in the index and find the matching pointers. Conceptually speaking, the index would resemble the presentation depicted in Figure below.

FIGURE Components of an index



note that the first PAINTER_NUM index key value (123) is found in records 1, 2, and 4 of the PAINTING table. The second PAINTER_NUM index key value (126) is found in records 3 and 5 of the PAINTING table. DBMSs use indexes for many different purposes. You just learned that an index can be used to retrieve data more efficiently. But indexes can also be used by a DBMS to retrieve data ordered by a specific attribute or attributes. For example, creating an index on a customer's last name will allow you to retrieve the customer data alphabetically by the customer's last name. Also, an index key can be composed of one or more attributes. For example, in Figure 3.30, you can create an index on VEND_CODE and PROD_CODE to retrieve all rows in the PRODUCT table ordered by vendor, and within vendor, ordered by product. Indexes play an important role in DBMSs for the implementation of primary keys. When you define a table's primary key, the DBMS automatically creates a unique index on the primary key column(s) you declared. For example, in Figure 3.30, when you declare CUS_CODE to be the primary key of the CUSTOMER table, the DBMS automatically creates a unique index on that attribute. A **unique index**, as its name implies, is an index in which the index key can have only one pointer value (row) associated with it. (The index in Figure 3.32 is not a unique index because the PAINTER_NUM has multiple pointer values associated with it. For example, painter number 123 points to three rows—1, 2, and 4—in the PAINTING table.) A table can have many indexes, but each index is associated with only one table. The index key can have multiple attributes (composite index). Creating an index is easy. You will learn that a simple SQL command produces any required index.

3.9 Codd's Relational Database Rules

In 1985, Dr. E. F. Codd published a list of 12 rules to define a relational database system.2 The reason Dr. Codd published the list was his concern that many vendors were marketing products as "relational" even though those products did not meet minimum relational standards. Dr. Codd's list, shown in Table 3.8, serves as a frame of reference for what a truly relational database should be. Bear in mind that even the dominant database vendors do not fully support all 12 rules.

TABLE **Dr. Codd's 12 Relational Database Rules**

RULE	RULE NAME	DESCRIPTION
1	**Information**	All information in a relational database must be logically represented as column values in rows within tables.
2	**Guaranteed Access**	Every value in a table is guaranteed to be accessible through a combination of table name, primary key value, and column name.
3	**Systematic Treatment of Nulls**	Nulls must be represented and treated in a systematic way, independent of data type.
4	**Dynamic Online Catalog Based on the Relational Model**	The metadata must be stored and managed as ordinary data, that is, in tables within the database. Such data must be available to authorized users using the standard database relational language.
5	**Comprehensive Data Sublanguage**	The relational database may support many languages. However, it must support one well-defined, declarative language with support for data definition, view definition, data manipulation (interactive and by program), integrity constraints, authorization, and transaction management (begin, commit, and rollback).
6	**View Updating**	Any view that is theoretically updatable must be updatable through the system.
7	**High-Level Insert, Update, and Delete**	The database must support set-level inserts, updates, and deletes.
8	**Physical Data Independence**	Application programs and ad hoc facilities are logically unaffected when physical access methods or storage structures are changed.
9	**Logical Data Independence**	Application programs and ad hoc facilities are logically unaffected when changes are made to the table structures that preserve the original table values (changing order of columns or inserting columns).
10	**Integrity Independence**	All relational integrity constraints must be definable in the relational language and stored in the system catalog, not at the application level.
11	**Distribution Independence**	The end users and application programs are unaware and unaffected by the data location (distributed vs. local databases).
12	**Nonsubversion**	If the system supports low-level access to the data, there must not be a way to bypass the integrity rules of the database.
	Rule Zero	All preceding rules are based on the notion that in order for a database to be considered relational, it must use its relational facilities exclusively to manage the database.

Summary

◗ Tables are the basic building blocks of a relational database. A grouping of related entities, known as an entity set, is stored in a table. Conceptually speaking, the relational table is composed of intersecting rows (tuples) and columns. Each row represents a single entity, and each column represents the characteristics (attributes) of the entities.

◗ Keys are central to the use of relational tables. Keys define functional dependencies; that is, other attributes are dependent on the key and can, therefore, be found if the key value is known. A key can be classified as a superkey, a candidate key, a primary key, a secondary key, or a foreign key.

◗ Each table row must have a primary key. The primary key is an attribute or a combination of attributes that uniquely identifies all remaining attributes found in any given row. Because a primary key must be unique, no null values are allowed if entity integrity is to be maintained. ◗ Although the tables are independent, they can be linked by common attributes. Thus, the primary key of one table can appear as the foreign key in another table to which it is linked. Referential integrity dictates that the foreign key must contain values that match the primary key in the related table or must contain nulls.

◗ The relational model supports relational algebra functions: SELECT, PROJECT, JOIN, INTERSECT, UNION, DIFFERENCE, PRODUCT, and DIVIDE. A relational database performs

much of the data manipulation work behind the scenes. For example, when you create a database, the RDBMS automatically produces a structure to house a data dictionary for your database. Each time you create a new table within the database, the RDBMS updates the data dictionary, thereby providing the database documentation.

◗ Once you know the relational database basics, you can concentrate on design. Good design begins by identifying appropriate entities and their attributes and then the relationships among the entities. Those relationships (1:1, 1:M, and M:N) can be represented using ERDs. The use of ERDs allows you to create and evaluate simple logical design. The 1:M relationship is most easily incorporated in a good design.

Review Questions

1. What is the difference between a database and a table?
2. What does it mean to say that a database displays both entity integrity and referential integrity?
3. Why are entity integrity and referential integrity important in a database?
4. What are the requirements that two relations must satisfy in order to be considered unioncompatible?
5. Which relational algebra operators can be applied to a pair of tables that are not unioncompatible?
6. Explain why the data dictionary is sometimes called “the database designer’s database.”
7. A database user manually notes that “The file contains two hundred records, each record containing nine fields.” Use appropriate relational database terminology to “translate” that statement.