Learner Guide

Faculty of Information Technology

Data Base Systems 600

Year 2 *Semester 1*

RICHFIELD

richfield.ac.za

Registered with the Department of Higher Education as a Private Higher Education Institution under the Higher Education Act, 1997.
Registration Certificate No. 2000/HE07/008

FACULTY OF INFORMATION TECHNOLOGY

LEARNER GUIDE

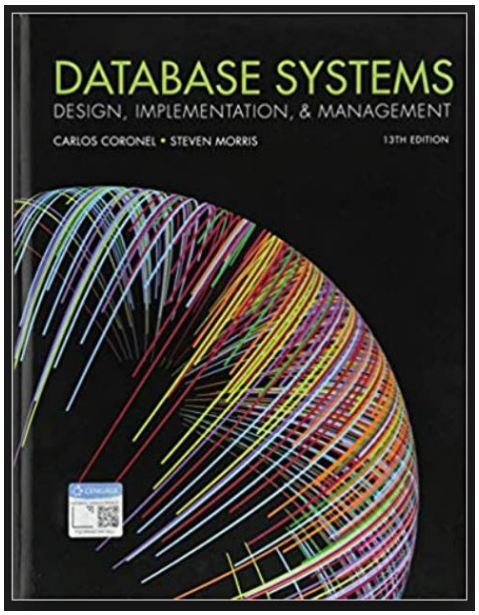
MODULE: DATA BASE SYSTEMS 600 (1ST SEMESTER)

PREPARED ON BEHALF OF
RICHFIELD GRADUATE INSTITUTE OF TECHNOLOGY (PTY) LTD

GRADUATE INDTITUTE OF TECHNOLOGY (PTY) LTD
Registration Number: 2000/000757/07

Contents

PRESCRIBED OR RECOMMENDED BOOKS



Database Systems: Design, Implementation, and Management, 13th Edition Carlos Coronel and Steven Morris

ISBN-13: 978-1337627900
ISBN-10: 1337627909

Chapter 1: Intro to Database Concepts

Chapter 2: Relational Database Modelling

Chapter 3: Normalisations

Chapter 4: Working with SQL

Chapter 5: Database Design

Chapter 6: Performance Tuning

Chapter 7: Database Connectivity and Internet Technologies

Chapter 8: Intro to Database Administration

LEARNING OUTCOMES

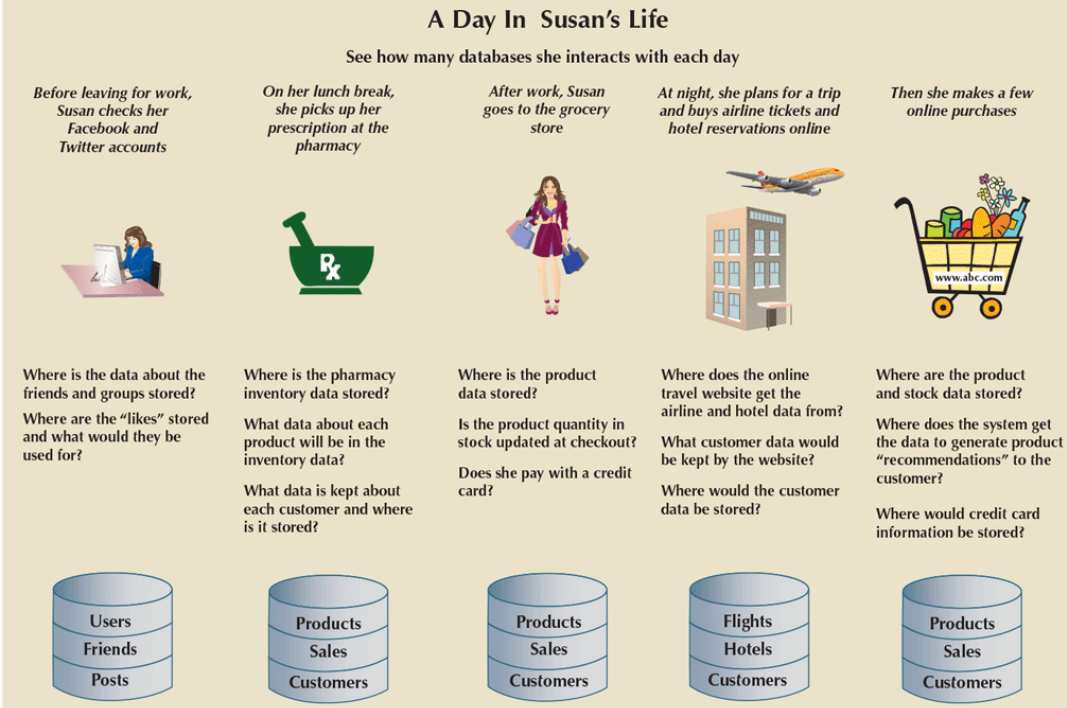
After reading this Section of the guide, the learner should be able to:

- Discuss the terms data and information as well as differentiate between them
- Give a comprehensive description of what a database is, the various types of databases, and discuss their value as an asset for decision making
- Understand and provide a detailed explanation of the importance of database design
- Be well acquainted with the evolution of modern databases from file systems
- Understand flaws in file system data management
- Outline the main components of the database system
- Give a full description of the main functions of a database management system (DBMS)
- Discuss data modelling, its building blocks and the importance of data models
- Understand and define what business rules are and how they influence database design

Why Databases?

So, why do we need databases? In today's world, data is ubiquitous (abundant, global, every- where), pervasive (unescapable, prevalent, persistent) and essential for organizations to survive and prosper. From birth to death, we generate and consume enormous amounts of data, starting with the birth certificate and continuing all the way to a death certificate. As you will learn in this module, databases are the best way to store and manage data. Simply put, databases make data persistent and shareable in a secure way. As you look at Figure 1.1, can you identify some of the data generated by your own daily activities?

FIGURE 1.1 THE PERVASIVE NATURE OF DATABASES



Telecommunications companies, such as Sprint and AT&T, are known to have systems that keep data on trillions of phone calls, with new data being added to the system at speeds up to 70,000 calls per second! Not only do such companies have to store and manage immense collections of data but they must be able to find any given fact in that data quickly. How can these businesses process this much data? How can they store it all, and then quickly retrieve just the facts that decision makers want to know, just when they want to know it? The answer is that they use databases. Databases, as explained in detail throughout this book, are specialized structures that allow computer-based systems to store, manage, and retrieve data very quickly. Virtually all modern business systems rely on databases. Therefore, a good understanding of how these structures are created and their proper use is vital for any information systems professional.

Data versus Information

To understand what drives database design, you must understand the difference between data and information. **Data** consists of raw (unprocessed) facts. For example, suppose that a university tracks data on faculty members for reporting to accrediting bodies. To get the data for each faculty member into the database, you would provide a screen to allow for convenient data entry, complete with all

the necessary data-entry validation controls. When the data is entered into the form and saved, it is placed an underlying database as raw data. Although you now have the facts in hand, reading through umpteen rows of data for faculty members does not provide much insight into the overall makeup of the faculty. Therefore, you transform the raw data into a data summary from which you can get quick answers to questions that are critical to decision making.

Information is the result of processing raw data to reveal its meaning. Data processing can be as simple as organizing data to reveal patterns or as complex as making forecasts or drawing inferences using statistical modelling. To reveal meaning, information requires context. Then it can be used as the foundation for decision making. For example, the data summary for the faculty can provide accrediting bodies with insights that are useful in determining whether to renew accreditation for the university.

In this "information age," production of accurate, relevant, and timely information is the key to good decision making. In turn, good decision making is the key to business survival in a global market. Data is the foundation of information, which is the bedrock of knowledge. Knowledge implies familiarity, awareness, and understanding of information as it applies to an environment.

Above we have explained the importance of data, and how the processing of data is used to reveal information which generates "actionable" knowledge. Let's explore a simple example of how this works in the real world. In today's information-centric society, we use smartphones daily. These devices have advanced GPS functionality that constantly tracks your whereabouts. This data is stored and shared with various applications. In some cases, the information generated is very helpful: it can help you navigate to various locations and even to find where you parked your car.

As you can see from this example, knowledge and information require timely and accurate data. Such data must be properly generated and stored in a format that is easy to access and process. **Data management** is a discipline that focuses on the proper generation, storage, and retrieval of data. Given the crucial role that data plays, it should not surprise you that it is a core activity for any business, government agency, service organization, or charity.

Read

Database Systems: Design, Implementation, and Management, 13th Edition Carlos Coronel and Steven Morris, Chapter One, page 4-6.

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 the following:

- End-user data—that is, raw facts of interest to the end user
- Metadata, or data about data, describes the data characteristics and relationships linking the data

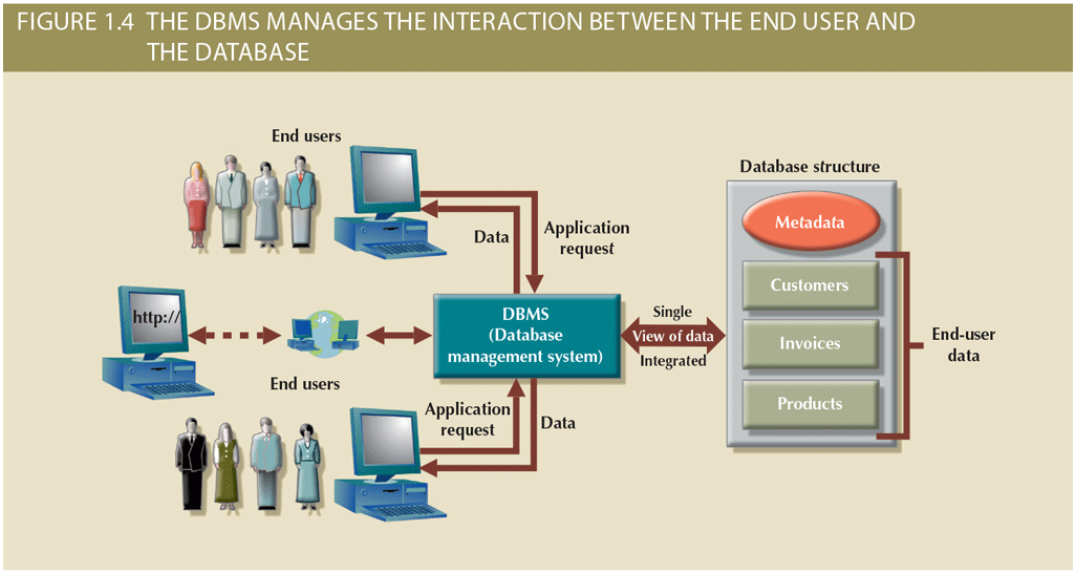
The metadata component, for example, stores information such as the name of each data element, the type of values (numeric, dates, or text) stored on each data element, and whether the data element can be left empty or not. This provides information that complements and expands the value and use of the 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. In a sense, a database resembles a very well-organized electronic filing cabinet in which powerful software (the DBMS) helps manage the cabinet's contents.

The role and advantages of a 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. Figure 1.4 emphasizes the point that the DBMS presents the end user 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 fulfil those requests. It hides much of the database's internal complexity from the application programs and users.

FIGURE 1.4 THE DBMS MANAGES THE INTERACTION BETWEEN THE END USER AND THE DATABASE



Having a DBMS between the end user's applications and the database offers important advantages. Some of which are:

- *Improved data sharing and access*. It helps create an environment which makes it possible: for end users have better access to more and better-managed data, and to produce quick answers to **ad hoc queries** (spur-of-the-moment requests to the DBMS for data manipulation).
- *Improved data security*. 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.
- *Minimized data inconsistency*. **Data inconsistency** exists when different versions of the same data appear in different places. The probability of data inconsistency is greatly reduced in a properly designed database.
- *Improved decision making*. The DBMS provides a framework to facilitate **data quality** (a comprehensive approach to promoting the accuracy, validity, and timeliness of data) initiatives. Data quality concepts are covered in more detail in Chapter 16 of the prescribed textbook, 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.

Types of Databases

A DBMS can be used to build many different types of databases. Each database stores a collection of data and is used for a specific purpose and different methods have been used to classify these databases.

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. 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 and can be either classified as a **workgroup database** or 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 are addressed in detail in Chapter 12, Distributed Database Management Systems.

In recent years, the use of cloud databases has been growing in popularity. A **cloud database** is a database that is created and maintained using cloud data services, such as Microsoft Azure or Amazon AWS.

In some contexts, such as research environments, a popular way of classifying data- bases is according to the type of data stored in them. Using this criterion, databases are grouped into two categories: general-purpose and discipline-specific databases. **General-purpose databases** contain a wide variety of data used in multiple disciplines—for example, the LexisNexis and ProQuest databases that contain newspaper, magazine, and journal articles for a variety of topics. **Discipline-specific databases** contain data focused on specific subject areas which is used for academic or research purposes. An example of a discipline-specific database is the geographic information system (GIS) database that store geospatial and other related data.

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. A database that is designed primarily to support a company's day-to-day operations is classified as **an operational database**, also known as an **online transaction processing (OLTP) database**, **transactional database**, or **production database**. In contrast, an **analytical database** focuses primarily on storing historical data and business metrics used exclusively for tactical or strategic decision making.

Typically, analytical databases comprise two main components: a data warehouse and an online analytical processing front end. The **data warehouse** is a specialized database that stores data in a format optimized for decision support. **Online analytical processing (OLAP)** is a set of tools that work together to provide an advanced data analysis environment for retrieving, processing, and modelling data from the data warehouse. In recent times, this area of database application has grown in importance and usage, to the point that it has evolved into its own discipline: **business intelligence**.

Databases can also be classified to reflect the degree to which the data is structured. **Unstructured data** is data that exists in the format in which it was collected which does not lend itself to the

processing that yields information. **Structured data** is the result of formatting unstructured data to facilitate storage, use, and generation of information. You apply structure (format) based on the type of processing that you intend to perform on the data.

Most data you encounter is best classified as semi-structured. **Semi-structured data** has already been processed to some extent. For example, if you look at a typical webpage, the data is presented 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. Unstructured and semi-structured data storage and management needs are being addressed through a new generation of databases known and as XML databases. **Extensible Mark-up 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. Table 1.1 compares the features of several well-known database management systems.

TABLE 1.1

TYPES OF DATABASES

PRODUCT	NUMBER OF USERS			DATA LOCATION		DATA USAGE		XML
	SINGLE USER	MULTIUSER						
		WORKGROUP	ENTERPRISE	CENTRALIZED	DISTRIBUTED	OPERATIONAL	ANALYTICAL	
MS Access	X	X		X		X		
MS SQL Server	X*	X	X	X	X	X	X	X
IBM DB2	X*	X	X	X	X	X	X	X
MySQL	X	X	X	X	X	X	X	X
Oracle RDBMS	X*	X	X	X	X	X	X	X

* Vendor offers single-user/personal or Express DBMS versions

With the emergence of the web and Internet-based technologies as the basis for the new "social media" generation, great amounts of data are being stored and analysed. Websites such as Google, Facebook, Twitter, and LinkedIn capture vast amounts of data about end users and consumers. This data grows exponentially and requires the use of specialized database systems. Over the past few years, this new breed of specialized database has grown in sophistication and wide-spread usage. This new type of database is known as a NoSQL database. The term **NoSQL** (Not only SQL) is generally used to describe a new generation of DBMS that is not based on the traditional relational database model. NoSQL databases are designed to handle the unprecedented volume of data, variety of data types and structures, and velocity of data operations that are characteristic of these new business requirements. You will learn more about this type of system in Chapter 2, Data Models.

Read

Database Systems: Design, Implementation, and Management, 13th Edition Carlos Coronel and Steven Morris, Chapter One, page 6-12.

Why Database Design is Important

A problem that has evolved with the use of personal productivity tools such as spreadsheets and desktop database programs is that users typically lack proper data-modelling and database design skills. People naturally have a "narrow" view of the data in their environment which results in a poor database design.

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. It is such a crucial aspect of working with databases that most of the content of this module is dedicated to the development of good database design techniques. Even a good DBMS will perform poorly with a badly designed database.

Data is one of an organization's most valuable assets. Data on customers, employees, orders, and receipts is all vital to the existence of a company. Tracking key growth and performance indicators are also vital to strategic and tactical plans to ensure future success; therefore, an organization's data must not be handled lightly or carelessly. Thorough planning to ensure that data is properly used and leveraged to give the company the most benefit is just as important as proper financial planning to ensure that the company gets the best use from its financial resources.

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 module focuses on the design of transactional, centralized, single-user, and multiuser databases.

FIGURE 1.6 EMPLOYEE SKILLS CERTIFICATION IN A GOOD DESIGN

Table name: EMPLOYEE

Employee_ID	Employee_FName	Employee_LName	Employee_HireDate	Employee_Title
02345	Brian	Oates	2/14/1999	DBA
03373	Franklin	Johnson	3/15/2006	Purchasing Agent
04893	Patricia	Richards	6/11/2008	DBA
06234	Jasmine	Patel	8/10/2009	Programmer
08273	Marco	Bienz	7/28/2010	Analyst
09002	Wade	Gaither	5/20/2014	Clerk
09283	Juan	Chavez	7/4/2014	Clerk
09382	Susan	Mathis	8/2/2014	Database Programmer
10282	Amanda	Richardson	4/11/2015	Clerk
13383	Raymond	Matthews	3/12/2016	Programmer
13567	Robert	Almond	9/30/2016	Analyst
13932	Megan	Lee	9/29/2017	Programmer
14311	Lee	Duong	9/1/2018	Programmer

Database name: Ch01_Text

Table name: CERTIFIED

Employee_ID	Skill_ID	Certified_Date
02345	100	2/14/2004
02345	110	8/9/2005
02345	180	2/14/2007
03373	120	6/20/2013
04893	180	6/11/2008
04893	220	9/20/2014
06234	110	8/10/2009
06234	200	8/10/2009
06234	210	1/29/2014
08273	110	3/8/2011
08273	190	8/19/2014
09002	110	5/16/2015
09002	120	5/16/2015
09382	140	8/2/2014
09382	210	8/2/2014
09382	220	5/1/2015
13383	170	3/12/2016
13567	130	9/30/2016
13567	140	5/23/2017
14311	110	9/1/2018

Table name: SKILL

Skill_ID	Skill_Name	Skill_Description
100	Basic Database Management	Create and manage database user accounts.
110	Basic Web Design	Create and maintain HTML and CSS documents.
120	Advanced Spreadsheets	Use of advanced functions, user-defined functions, and macroing.
130	Basic Process Modeling	Create core business process models using standard libraries.
140	Basic Database Design	Create simple data models.
150	Master Database Programming	Create integrated trigger and procedure packages for a distributed environment.
160	Basic Spreadsheets	Create single tab worksheets with basic formulas
170	Basic C# Programming	Create single-tier data aware modules.
180	Advanced Database Management	Manage Database Server Clusters.
190	Advance Process Modeling	Evaluate and Redesign cross-functional internal and external business processes.
200	Advanced C# Programming	Create multi-tier applications using multi-threading
210	Basic Database Manipulation	Create simple data retrieval and manipulation statements in SQL.
220	Advanced Database Manipulation	Use of advanced data manipulation methods for multi-table inserts, set operations, and correlated subqueries.

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 poor decision making which can lead to the failure of an organization. Hence, 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.

Read

Database Systems: Design, Implementation, and Management, 13th Edition Carlos Coronel and Steven Morris, Chapter One, page 12-15

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 help 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, they only 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 these problems and how to avoid them.

Manual File Systems

To be successful, an organization must develop systems for handling core business tasks. Historically, such systems were often manual, paper-and-pen based and organized in a system of file folders and filing cabinets, to facilitate the expected use of the data. If a collection of data 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.

Computerised File Systems

Generating reports from manual file systems was slow and cumbersome. 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 like the manual files and were thus unsatisfactory for a database.

The description of computer files requires a specialized vocabulary. Every discipline develops its own terminology to enable its practitioners to communicate clearly. The basic file vocabulary shown in Table 1.2 will help you to understand subsequent discussions more easily.

TABLE 1.2

BASIC FILE TERMINOLOGY

TERM	DEFINITION
Data	Raw facts, such as a telephone number, a birth date, a customer name, and a year-to-date (YTD) sales value. Data has little meaning unless it has been organized in some logical manner.
Field	A character or group of characters (alphabetic or numeric) that has a specific meaning. A field is used to define and store data.
Record	A logically connected set of one or more fields that describes a person, place, or thing. For example, the fields that constitute a record for a customer might consist of the customer's name, address, phone number, date of birth, credit limit, and unpaid balance.
File	A collection of related records. For example, a file might contain data about the students currently enrolled at Gigantic University.

When business users wanted data from the computerized file, they sent requests for the data to the DP specialist. For each request, the DP specialist had to create reusable programs to retrieve the data from the file, manipulate it in whatever manner the user had requested, and present it as a printed report. The demand for different reports increased and generated more requests for the DP specialist to create more computerized files of other business data, which in turn meant that more data management programs had to be created, which led to even more requests for reports.

As more and more computerized files were developed, the problems with this type of file system became apparent. While these problems are explored in detail in the next section, the problems basically centred on having many data files that contained related—often overlapping—data with no means of controlling or managing the data consistently across all the files. Each file in the system used its own application program to store, retrieve, and modify data. Furthermore, each file was owned by an individual or a department that commissioned its creation.

The advent of computer files to store company data was significant; as it represented a huge step forward in a business's ability to process data. Previously, users had direct, hands-on access to all of the business data. But they didn't have the tools to convert that data into the information they needed. The creation of computerized file systems gave them improved tools for manipulating the company data that allowed them to create new information. However, it had the additional effect of introducing a schism between the end users and their data. The desire to close the gap between the end users and the data influenced the development of many types of computer technologies, system designs, and uses (and misuses) of many technologies and techniques. However, such developments also created a split between the ways DP specialists and end users viewed the data.

File System Redux: Modern End-User Productivity Tools

The users' desire for direct, hands-on access to data helped to fuel the adoption of personal computers for business use. However, the ubiquitous use of personal productivity tools can introduce the same problems as the old file systems.

Personal computer spreadsheet programs such as Microsoft Excel are widely used by business users, and they allow the user to enter data in a series of rows and columns, so the data can be manipulated using a wide range of functions. The popularity of spreadsheet applications has enabled users to conduct sophisticated data analysis that has greatly enhanced their ability to understand the data and make better decisions. Unfortunately, users have become so adept at working with spreadsheets that they tend to use them to complete tasks for which spreadsheets are not appropriate. A common misuse of spreadsheets is as a substitute for a database. Due to the large number of users with spreadsheets, each making separate copies of the data, the resulting "file system" of spreadsheets suffers from the same problems as the file systems created by the early DP specialists, which are outlined in the next section.

Read

Database Systems: Design, Implementation, and Management, 13th Edition Carlos Coronel and Steven Morris, Chapter One, page 15-18.

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. Nonetheless, many problems and limitations became evident in this approach.

The following problems associated with file systems, 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 simplest data-retrieval task requires extensive programming.
- Difficulty of getting quick answers. The need to write programs to produce even the simplest reports makes ad hoc queries impossible. Harried DP specialists often received numerous requests for new reports which were produced, a week or a month later. 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. Because ad hoc queries are not possible, the file reporting programs can multiply quickly, and 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. 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, they are insufficient for robust data sharing among users.
- Extensive programming. Making changes to an existing file structure can be difficult in a file system environment. 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.

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 a CUSTOMER file would require the four steps described in the previous section resulting in the need for the file system programs to be modified to conform to the new file structure. Thus, the file system application programs exhibit structural dependence as well as **data dependence**.

The practical significance of data dependence is the difference between the **logical data format** and the **physical data format**. 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.

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. Because data stored in different locations will probably not be updated consistently, the **islands of information** often contain different versions of the same data, producing data redundancy. **Data redundancy** exists when the same data is stored unnecessarily at different places. This sets the stage for the following:

- ***Poor data security***. Having multiple copies of data increases the chances for a copy of the data to be susceptible to unauthorized access.
- ***Data inconsistency***. Data inconsistency exists when different and conflicting versions of the same data appear in different places.
- ***Data-entry errors***. Data-entry errors are more likely to occur when complex entries (such as 10-digit phone numbers) are made in several different files or recur frequently in one or more files.
- ***Data integrity problems***. It is possible to enter a non-existent 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.

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. If agent Leah F. Hahn, stored in a CUSTOMER file, decides to get married and move, the agent's name, address, and phone number are likely to change, and you could be faced with the prospect of making hundreds of corrections as any change in any field value must be correctly made in many places to maintain data integrity. A data anomaly develops when not all the required changes in the redundant data are made successfully. The data anomalies possible in this case are commonly defined as follows:

- ***Update anomalies***. If agent Leah F. Hahn has a new phone number, it must be entered in each of the CUSTOMER file records in which Ms. Hahn's phone number is shown. 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, and you needed to add a new agent, you would also add a dummy customer data entry to reflect the new agent's addition and the potential for creating data inconsistencies would be great.
- ***Deletion anomalies***. If you delete 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.

On a positive note, however, this book will help you develop the skills needed to design and model a successful database that avoids the problems listed in this section.

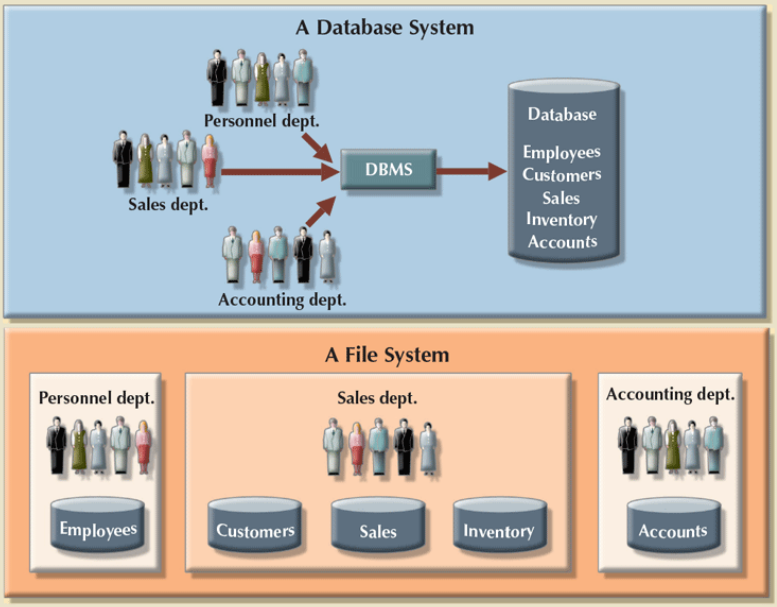
Read

Database Systems: Design, Implementation, and Management, 13th Edition Carlos Coronel and Steven Morris. Chapter One. page 18-21.

Database Systems

The problems inherent in file systems make using a database system very desirable. The database system consists of logically related data stored in a single logical data repository. "Logical" because the data repository appears to be a single unit to the end user, even though data might be physically distributed among multiple storage facilities and locations. The database's DBMS, shown in Figure 1.10, provides numerous advantages over file system management, by making it possible to eliminate most of the file system's problems. Better yet, the current generation of DBMS software stores not only the data structures but also the relationships between those structures in a central location and takes care of defining, storing, and managing all required access paths to those components. Remember, the DBMS is just one of several crucial components of a database system.

FIGURE 1.10 CONTRASTING DATABASE AND FILE SYSTEMS



However, 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 picture.

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, namely:

- ***Hardware***. The system's physical devices, including computers, storage devices, and other devices.
- ***Software***. Three types of software are needed to make the database system function fully. These are listed and described below:
    - *Operating system software*, such as Microsoft Windows, Linux and Mac OS, manages all hardware components and makes it possible for all other software to run on the computers.
    - *DBMS software* manages the database within the database system. Some examples of DBMS software are Microsoft's SQL Server, Oracle Corporation's Oracle, Oracle'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. For example, 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. Based on primary job functions, five types of users can be identified in a database system. Each user type, described next, 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.
  - *Database designers* design the database structure. They are, in effect, the database architects.
  - *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.

- ***Procedures***. Procedures are the instructions and rules that govern the design and use of the database system. Procedures are a critical component of the system because they enforce the standards by which business is conducted within the organization and with customers. Procedures also help to ensure that companies have an organized way to monitor and audit the data that enter the database and the information generated from those data.
- ***Data***. The word data covers the collection of facts stored in the database. Determining which data to enter into the database and how to organize that data is a vital part of the database designer's job.

A database system adds a new dimension to an organization's management structure. The complexity of this managerial structure depends on the organization's size, its functions, and its corporate culture. Therefore, database systems can be created and managed at different levels of complexity and with varying adherence to precise standards. In addition to the different levels of database system complexity, managers must also take another important fact into account: database solutions must be cost-effective as well as tactically and strategically effective. Producing a million-dollar solution to a thousand-dollar problem is hardly an example of good database system selection or of good database design and management. Finally, the database technology already in use is likely to affect the selection of a database system.

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 can be achieved only through a DBMS. Each of these functions is explained as follows:

- *Data dictionary management*. The DBMS stores definitions of the data elements and their relationships (metadata) in a data dictionary which also automatically records structural changes. 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. This provides data abstraction and removes structural and data dependence from the system.

- *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 also provides storage for related data-entry forms or screen definitions, report definitions, data validation rules, 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.
- *Data transformation and presentation*. The DBMS transforms entered data to conform to required data structures. That is, the DBMS formats the physically retrieved data to make it conform to the user's logical expectations.
- *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.
- *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 its integrity.
- *Backup and recovery management*. The DBMS provides backup and data recovery to ensure data safety and integrity. Current DBMS systems 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.
- *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.
- *Database access languages and application programming interfaces*. The DBMS provides data access through a **query language**. Structured Query Language (SQL) is the de facto query language and data access standard supported by majority of DBMS vendors. The DBMS also provides application programming interfaces to procedural languages such as COBOL, C, Java, Visual Basic.NET, and C#.
- *Database communication interfaces*. A current-generation DBMS accepts end-user requests via multiple, different network environments. In this environment, communications can be accomplished in several ways:
    - End users can generate answers to queries by filling in screen forms through a web browser.
    - The DBMS can automatically publish predefined reports on a website.
    - The DBMS can connect to third-party systems to distribute information via email or other productivity applications.

Managing the Database System

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 to a focus on the broader aspects of managing the organization's data resources and on the administration of the complex database software itself. Although the database system yields considerable advantages over previous data management approaches, database systems do carry significant disadvantages:

- *Increased costs*. Database systems require sophisticated hardware and software and highly skilled personnel which require maintaining and the cost for that can be substantial.
- *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.

- *Maintaining currency*. To maximize the efficiency of the database system, you must keep your system current through the performance of frequent updates and application of the latest patches and security measures to all components.
- *Vendor dependence*. Given the heavy investment in technology and personnel training, companies might be reluctant to change database vendors. Therefore, vendors are less likely to offer pricing point advantages to customers, and those customers might be limited in their choice of database system components.
- *Frequent upgrade/replacement cycles*. DBMS vendors frequently upgrade their product versions by adding new functionality. Some of these versions require hardware upgrades and staff training which may be costly.

Now that you know what a database and DBMS are, and why they are necessary, you are ready to begin developing your career as a database professional.

Read

Database Systems: Design, Implementation, and Management, 13th Edition Carlos Coronel and Steven Morris, Chapter One, page 21-28.

1.2 Data Models

Data Modelling and Data Models

Database design focuses on how the database structure will be used to store and manage end-user data. **Data modelling** refers to the process of creating a specific data model for a determined problem domain. A data model is a relatively simple representation, usually graphical, of more complex real-world data structures. 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 modelling is an iterative, progressive process that must be done with care.

Traditionally, database designers relied on good judgment to help them develop a good data model and this involved a lot of trial and error. Fortunately, database designers now make use of existing data-modelling constructs and powerful database design tools that substantially diminish the potential for errors in database modelling. In the following sections, you will learn how existing data models are used to represent real-world data and how the different degrees of data abstraction facilitate data modelling.

Read

Database Systems: Design, Implementation, and Management, 13th Edition Carlos Coronel and Steven Morris, Chapter Two, page 36-36.

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. The importance of data modelling cannot be overstated. Data constitutes the most basic information employed by a system. Applications are created to manage data and to help transform data into information, but data is viewed in different ways by different people. The different users and producers of data and information often reflect the fable of the blind people and the elephant: they all have different views. A view of the whole elephant is needed. A sound data environment requires an overall database blueprint based on an appropriate data model. 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.

Data Model Basic Building Blocks

The basic building blocks of all data models are entities, attributes, relationships, and constraints. An **entity** is a person, place, thing, or event about which data will be collected and stored. An entity represents a type of object in the real world, and each entity occurrence is unique and distinct. For example, a CUSTOMER entity would have many distinguishable customer occurrences, such as John Smith, Pedro Dinamita, and Tom Strickland. Entities may be physical objects and may also be abstractions.

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 number, 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. The following examples illustrate the distinctions among the three relationships.

- **One-to-many (1:M or 1..*) relationship**. A painter creates many different paintings, but each 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.
- **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.
- **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 labelled 1:1.

The preceding discussion identified each relationship in both directions; that is, relationships are bidirectional.

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, such as:

- 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 modelling.

Read

Database Systems: Design, Implementation, and Management, 13th Edition Carlos Coronel and Steven Morris, Chapter Two, page 36-38.

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 exist in an organization, how the data is used, and in what time frames it is used. 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.

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. These business rules are then used to define entities, attributes, relationships, and constraints.

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. Examples of business rules are as follows:

- A customer may generate many invoices.
- A training session cannot be scheduled for fewer than 10 employees or for more than 30 employees.

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. However, because perceptions differ, end users are sometimes a less reliable source. Therefore, it is necessary to verify end-user perceptions to ensure accuracy. The process of identifying and documenting business rules is essential to database design for several reasons:

- It helps to standardize the company's view of data.
- It can be a communication tool between users and designers.
- It allows the designer to understand the nature, role, and scope of the data.
- It allows the designer to understand business processes.

- It allows 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 modelled. However, they can be represented and enforced by application software.

Translating Business Rules to 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 the objects. As a rule, a noun in a business rule will translate into an entity in the model, and a verb (active or passive) that associates the 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 the following:

- 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.

To properly identify the relationship type, you should generally 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?

You will have many opportunities to determine the relationships between entities as you proceed through this course, and soon the process will become second nature.

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 it unique and distinguishable from other objects in the problem domain. Therefore, it is important to 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 or abbreviation of the entity in which it occurs. For example, in the CUSTOMER entity, the customer's credit limit may be called CUS_CREDIT_LIMIT. This will become increasingly important in later chapters when you learn about 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.

Read

Database Systems: Design, Implementation, and Management, 13th Edition Carlos Coronel and Steven Morris, Chapter Two, page 38-40.

The Evolution of Data Models

The quest for better data management has led to several models that attempt to resolve the previous model's critical shortcomings and to provide solutions to ever-evolving data management needs. 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.

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 1969. The model's basic logical structure is represented by an upside-down tree of levels.

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. 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:

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

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 marked a breakthrough for both users and designers. Its conceptual simplicity set the stage for a genuine database revolution.

The relational model's 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 two-dimensional structure 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.

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.

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 (as outlined in Chapter 1, in the DBMS Functions section).

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 the physical details, while the user sees

the relational database as a collection of tables in which data is stored. The user can manipulate and query the data in a way that seems intuitive and logical.

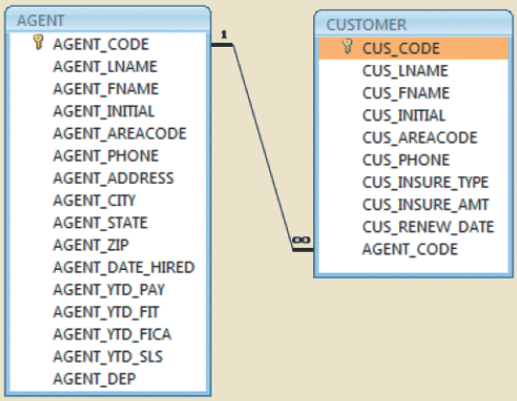
Tables are related to each other through the sharing of a common attribute (a value in a column).

Although the tables are independent of one another, you can easily associate the data between tables. The relational model provides a minimum level of controlled redundancy to eliminate most of the redundancies commonly found in file systems.

The relationship type (1:1, 1:M, or M:N) is often shown in a relational schema, an example of which is shown in Figure 2.2. A **relational diagram** is a representation of the relational database's entities, the attributes within those entities, and the relationships between those entities.

In Figure 2.2, the relational diagram shows the connecting fields and the relationship type (1:M).

FIGURE 2.2 A RELATIONAL DIAGRAM



A relational table stores a collection of related entities. In this respect, the relational database table resembles a file, but there is a crucial difference between a table and a file: a table yields complete data and structural independence because it is a purely logical structure. This property of the relational data model, which is explored in depth in the next chapter, became the source of a real data- base revolution.

Another reason for the relational data model's rise to dominance is its powerful and flexible query language. Most relational database software uses Structured Query Language (SQL), which allows the user to specify what must be done without specifying how. The RDBMS uses SQL to translate user queries into instructions for retrieving the requested data. From an end-user perspective, any SQL-based relational database application involves three parts: a user interface, a set of tables stored in the database, and the SQL "engine", explained as follows:

- *The end-user interfaces*. Basically, the interface allows the end user to interact with the data (by automatically generating SQL code).
- *A collection of tables stored in the database*. In a relational database, all data is perceived to be stored in tables.
- *SQL engine*. Largely hidden from the end user, the SQL engine executes all queries, or data requests. Hence, SQL is said to be a declarative language that tells what must be done but not how.

Because the RDBMS performs some tasks behind the scenes, it is not necessary to focus on the physical aspects of the database. Instead, the following chapters concentrate on the logical portion of the relational database and its design.

The Entity Relationship Model

The conceptual simplicity of relational database technology triggered the demand for RDBMSs. This and other factors, have thus created the need for more effective database design tools.

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 modelling.

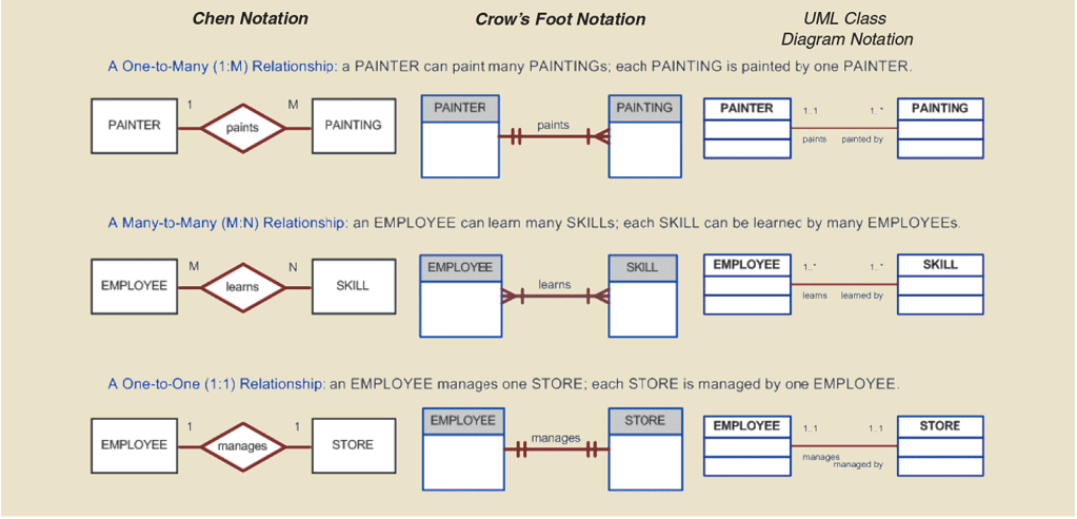
Peter Chen first introduced the ER data model in 1976. The relational data model and ERM combined to provide the foundation for tightly structured database design. ER models are normally represented in an **entity relationship diagram (ERD)**, which uses graphical representations to model database components.

The ER model is based on the following components:

- *Entity*. Earlier in this chapter, an entity was defined as anything about which data will 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 centre of (ER) model (ERM) the rectangle. The entity name is generally written in capital letters and in singular form. 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. A collection of like entities is known as an entity set. Technically speaking, the ERD (1:1, 1:M, and M:N) depicts entity sets.
- Each entity consists of a set of *attributes* that describes characteristics of the entity.
- *Relationships*. Relationships describe associations among data. Most relationships describe associations between two entities. The ER model uses the term **connectivity** to label the relationship types (1:1, 1:M, and M:N). The name of the relationship is usually an active or passive verb.

Figure 2.3 shows the different types of relationships using three ER notations: the original Chen notation, the Crow's Foot notation, and the newer class diagram notation, which is part of the Unified Modelling Language (UML).

FIGURE 2.3 THE ER MODEL NOTATIONS



The Crow's Foot notation is used as the design standard in this context. However, the Chen notation is used to illustrate some of the ER modelling concepts whenever necessary. Most data modelling tools let you select the Crow's Foot or UML class diagram notation.

ER model's exceptional visual simplicity makes it the dominant database modelling and design tool. Nevertheless, the search for better data-modelling tools continues as the data environment continues to evolve.

The Object- Oriented 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 its 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. An object is described by its factual content, 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.

Subsequent OODM development has allowed an object also to 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 relation- ships, and operational procedures, the object becomes self-contained, thus making it a basic building block for autonomous structures. The OO data model is based on the following components:

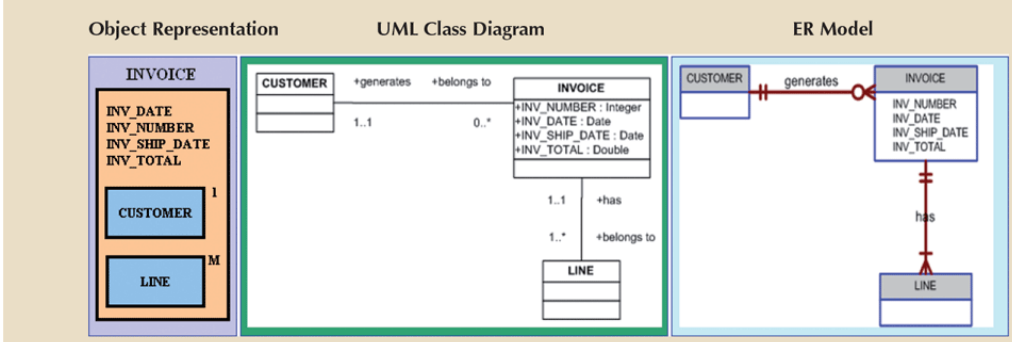
- An object is an abstraction of a real-world entity. In general terms, an object may be considered equivalent to an ER model's entity.
- Attributes describe the properties of an object. For example, a PERSON object includes the attributes Name, Social Security Number, and Date of Birth.
- Objects that share similar characteristics are grouped in classes. A **class** is a collection of similar objects with shared structure (attributes) and behaviour (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.
- 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.
- Object-oriented data models are typically depicted using **Unified Modelling Language (UML)** class diagrams. UML class diagrams are used to represent data and its relationships within the larger UML object-oriented system's modelling language.

To illustrate the main concepts of the OODM, consider a simple invoicing problem. In this case, invoices are generated by customers, each invoice references one or more lines, and each line represents an item purchased by a customer. Figure 2.4 illustrates the object representation for this simple invoicing problem, as well as the equivalent UML class diagram and ER model. The object representation is a simple way to visualize a single object occurrence.

FIGURE 2.4 A COMPARISON OF THE OO, UML, AND ER MODELS



The OODM advances influenced many areas, from system modelling to programming. The added semantics of the OODM allowed for a richer representation of complex objects. This in turn enabled applications to support increasingly complex objects in innovative ways. As you will see in the next section, such evolutionary advances also affected the relational model.

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 gave birth to a new generation of relational databases that support 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)**.

Today, most relational database products can be classified as object/relational, and they represent the dominant market share of OLTP and OLAP database applications. The success of the O/R DBMSs can be attributed to the model's conceptual simplicity, data integrity, easy-to-use query language, high transaction performance, high availability, security, scalability, and expandability. From the start, the OO and relational data models were developed in response to different problems.

The use of complex objects received a boost with the Internet revolution. When organizations integrated their business models with the Internet, they realized its potential to be an effective business communication tool. Within this environment, **extensible Mark-up Language (XML)** emerged as the de facto standard for the efficient and effective exchange of structured, semi-structured, and unstructured data. At the same time, O/R DBMSs added support for XML-based documents within their relational data structure. Due to its robust foundation in broadly applicable

principles, the relational model is easily extended to include new classes of capabilities, such as objects and XML.

Although relational and object/relational databases address most current data processing needs, a new generation of databases has emerged to address some very specific challenges found in some Internet-era organizations.

Emerging Data Models: Big Data and NoSQL

Deriving usable business information from the mountains of web data that organizations have accumulated over the years has become an imperative need. Web data in the form of browsing patterns, purchasing histories, customer preferences, behaviour patterns, and social media data have inundated organizations with combinations of structured and unstructured data. In addition, mobile technologies, plus sensors of all types as well as other Internet and cellular-connected devices, have created new ways to automatically collect massive amounts of data in multiple formats. The amount of data being collected grows exponentially every day. According to some studies, the rapid pace of data growth is the top challenge for organizations, with system performance and scalability as the next biggest challenges. The need to manage and leverage all these converging trends (rapid data growth, performance, scalability, and lower costs) has triggered a phenomenon called "Big Data." **Big Data** refers to a movement to find new and better ways to manage large amounts of web- and sensor-generated data and derive business insight from it, while simultaneously providing high performance and scalability at a reasonable cost.

It seems to be Douglas Laney, a data analyst from the Gartner Group, who first described the basic characteristics of Big Data databases: volume, velocity, and variety, or the 3 Vs.

The 3 Vs framework illustrates what companies now know, that the amount of data being collected in their databases has been growing exponentially in size and complexity. Traditional relational databases are good at managing structured data but are not well suited to managing and processing the amounts and types of data being collected in today's business environment.

The problem is that the relational approach does not always match the needs of organizations with Big Data challenges.

There is no "one-size-fits-all" cure to data management needs. For some organizations, creating a highly scalable, fault-tolerant infrastructure for Big Data analysis could prove to be a matter of business survival. The business world has many examples of companies that leverage technology to gain a competitive advantage, and others that miss it.

Will broadcast television networks be successful in adapting to streaming services such as Hulu, AppleTV, and Roku? Partnerships and mergers will undoubtedly change the landscape of home entertainment as the industry responds to the changing technological possibilities. Will traditional news outlets be able to adapt to the changing news consumption patterns of the millennial generation?

Big Data analytics are being used to create new types of services by all types of companies. For example, Amazon originally competed with "big box" department stores as a low-cost provider. Amazon eventually leveraged storage and processing technologies to begin competing in streaming movie and music service, and more recently, it has leveraged Big Data to create innovative services like predictive shipping. Amazon has also been successful with the sales of products like Amazon Echo that use the Alexa service to perform natural language processing. These "constantly listening" devices are embedded in homes around the world, providing Amazon with unprecedented levels and types of data that it can analyse to improve existing services and support innovation in future services.

To create value from their previously unused Big Data stores, companies are using new Big Data technologies which allow them to process massive data stores of multiple formats in cost-effective ways. Some of the most speed, fault-tolerant frequently used Big Data technologies are:

- **Hadoop** is a Java-based, open-source, high-speed, fault-tolerant distributed storage and computational framework which uses low-cost hardware to create clusters of thousands of computer nodes to store and process data.
- **Hadoop Distributed File System (HDFS)** is a highly distributed, fault-tolerant file storage system designed to manage large amounts of data at high speeds. In order to achieve high throughput, HDFS uses the write-once, read many model. This means that once the data is written, it cannot be modified.
- **MapReduce** is an open-source application programming interface (API) that provides fast data analytics services. MapReduce distributes the processing of the data among thousands of nodes in parallel. MapReduce works with structured and non-structured data. Although MapReduce itself is viewed as fairly limited today, it defined the paradigm for how Big Data is processed.
- **NoSQL** is a large-scale distributed database system that stores structured and unstructured data in efficient ways.

With the potential of big gains derived from Big Data analytics, it is not surprising that some organizations are turning to emerging Big Data technologies, such as NoSQL databases, to mine the wealth of information hidden in mountains of web data and gain a competitive advantage.

NoSQL Databases Every time you search for a product on Amazon, send messages to friends in Facebook, watch a video on YouTube, or search for directions in Google Maps, you are using a NoSQL database. As with any new technology, the term NoSQL can be loosely applied to many different types of technologies. However, this chapter uses NoSQL to refer to a new generation of databases that address the specific challenges of the Big Data era and have the following general characteristics:

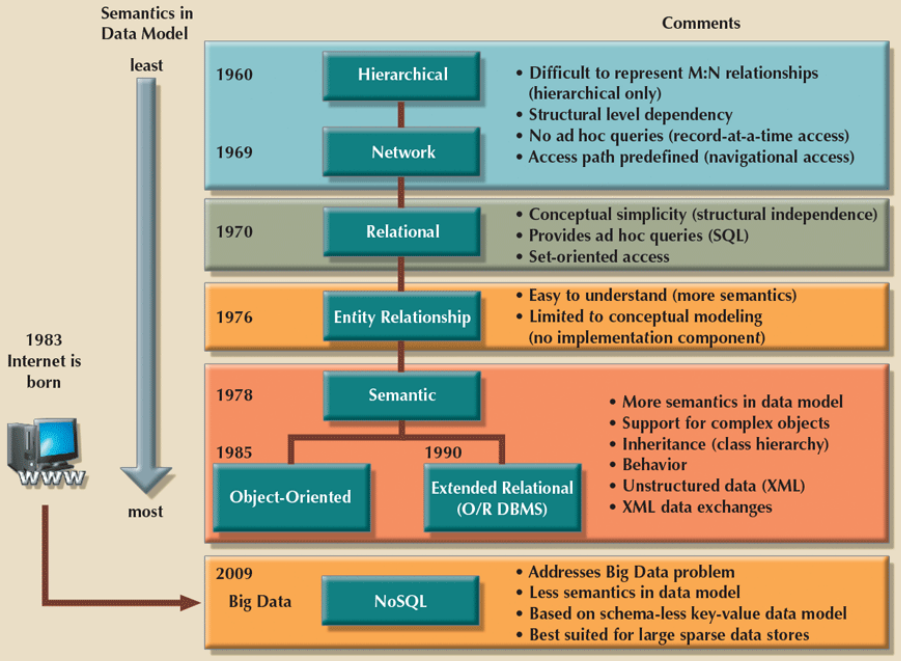
- They are not based on the relational model and SQL; hence the name NoSQL.
- They support highly distributed database architectures.
- They provide high scalability, high availability, and fault tolerance.
- They support very large amounts of sparse data.
- They are geared toward performance rather than transaction consistency.

Unlike the relational model, which provides a very comprehensive and cohesive approach to data storage and manipulation, the NoSQL model is a broad umbrella for a variety of approaches to data storage and manipulation.

Data Models: A Summary

The evolution of DBMSs has always been driven by the search for new ways of modelling and managing increasingly complex real-world data. A summary of the most commonly recognized data models is shown in Figure 2.5.

FIGURE 2.5 THE EVOLUTION OF DATA MODELS



In the evolution of data models, some common characteristics have made them widely accepted:

- A data model must show some degree of conceptual simplicity without compromising the semantic completeness of the database. A data model must represent the real world as closely as possible.
- Representation of the real-world transformations (behaviour) must be in compliance with the consistency and integrity characteristics required by the intended use of the 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 have 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 OO features to the relational model and allowed it to maintain strong market share within the business environment. In recent years, the Big Data phenomenon has stimulated the development of alternative ways to model, store, and manage data that represents a break with traditional data management. It is important to note that not all data models are created equal; some data models are better suited than others for some tasks.

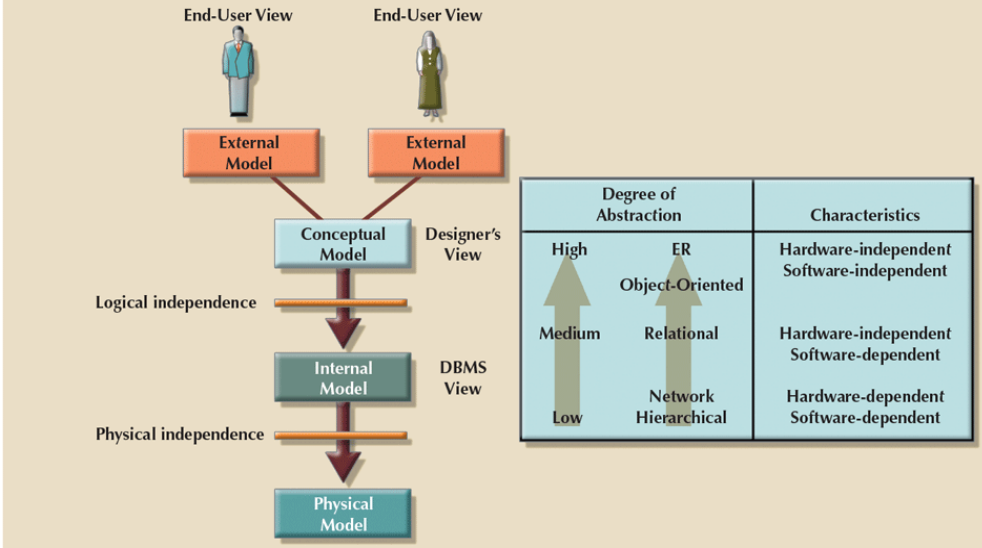
Read

Database Systems: Design, Implementation, and Management, 13th Edition Carlos Coronel and Steven Morris, Chapter Two, page 40-54

Degrees of Data Abstraction

If you ask 10 database designers what a data model is, you will end up with 10 different answers—depending on the degree of data abstraction. A database designer starts with an abstract view of the overall data environment and adds details as the design comes closer to implementation. Using levels of abstraction can also be very helpful in integrating multiple (and sometimes conflicting) views of data at different levels of an organization. In the early 1970s, the **American National standards institute (ANSI)** Standards Planning and Requirements Committee (SPARC) defined a framework for data modelling based on degrees of data abstraction. The resulting ANSI/SPARC architecture defines three levels of data abstraction: external, conceptual, and internal. You can use this framework to better understand database models, as shown in Figure 2.6. In the figure, the ANSI/SPARC framework has been expanded with the addition of a physical model to explicitly address physical-level implementation details of the internal model.

FIGURE 2.6 DATA ABSTRACTION LEVELS

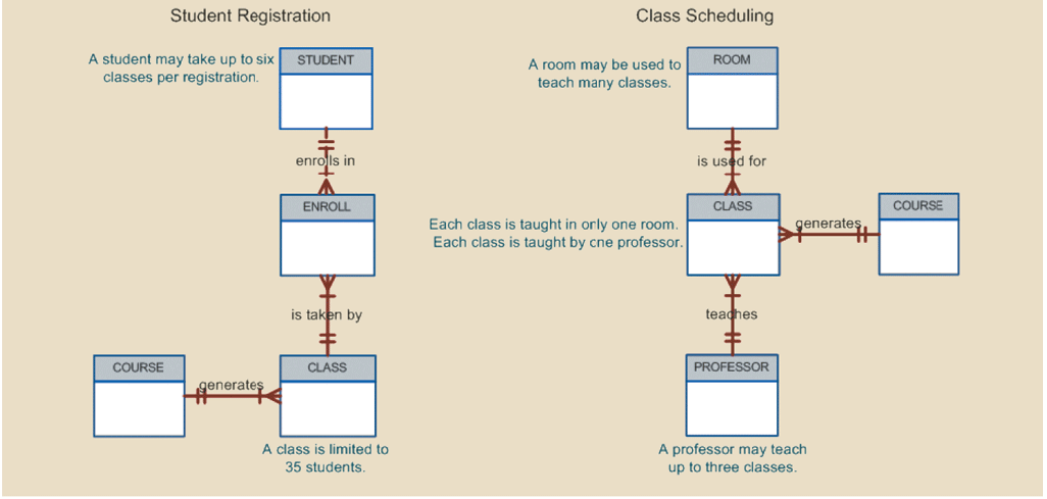


The External Model

The **external model** is the end users' view of the data environment. The term *end users* refers to people who use the application programs to manipulate the data and generate information. End users within different business units of a company view their data subsets as separate from or external to other units within the organization.

Because data is being modelled, ER diagrams will be used to represent the external views. A specific representation of an external view is known as an external schema (see example in Figure 2.7). 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.*

FIGURE 2.7 EXTERNAL MODELS FOR TINY COLLEGE



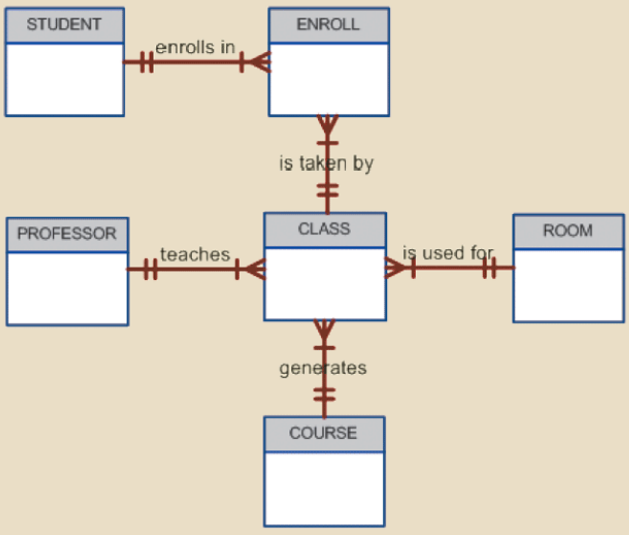
The use of external views that represent subsets of the database has some important advantages:

- It is easy to identify specific data required to support each business unit's operations.
- It makes the designer's job easy by providing feedback about the model's adequacy. Specifically, the model can be checked to ensure that it supports all processes as defined by their external models, as well as all operational requirements and constraints.
- It helps to ensure security constraints in the database design. Damaging an entire database is more difficult when each business unit works with only a subset of data.
- It makes application program development much simpler.

The Conceptual Model

The **conceptual model** represents a global view of the entire database 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, as shown in Figure 2.8. 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).

FIGURE 2.8 A CONCEPTUAL MODEL FOR TINY COLLEGE



The most widely used conceptual model is the ER model. With the ERD being used to graphically represent the **conceptual schema**. The conceptual model yields some important advantages. First, it provides a bird's- eye (macro level) view of the data environment that is relatively easy to understand.

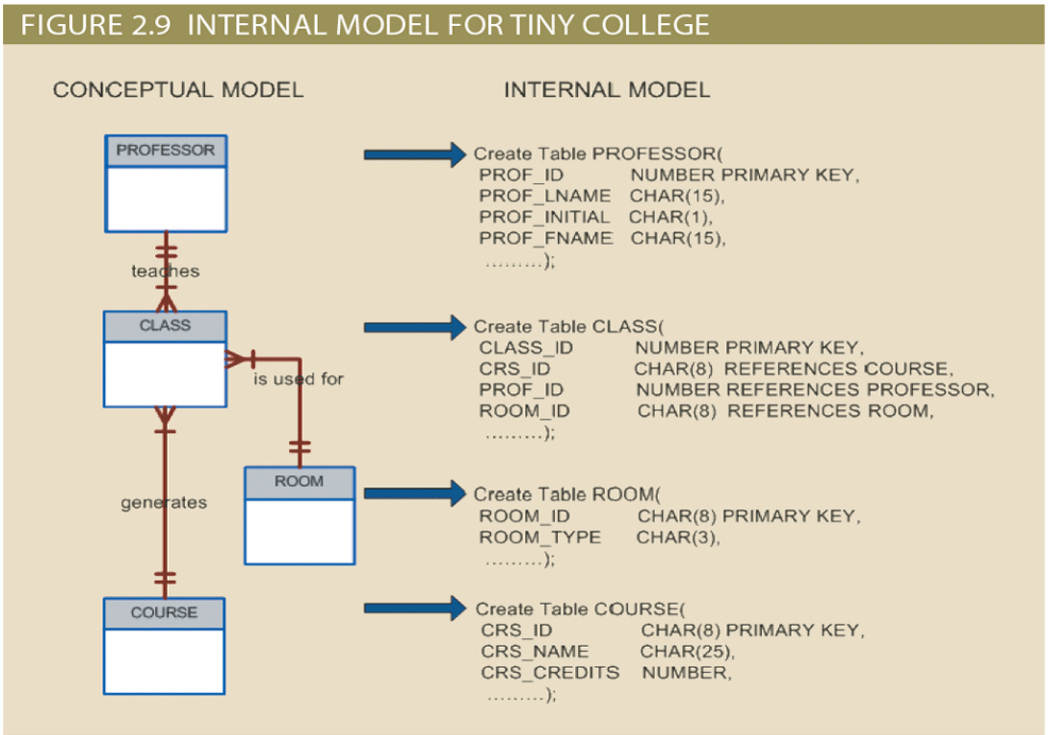
Second, the conceptual model is independent of both software and hardware. Therefore, changes in either the hardware or the DBMS software will have no effect on the database design at the conceptual level. Generally, the term **logical design** refers to the task of creating a conceptual data model that could be implemented in any DBMS.

The Internal Model

The **internal model** is the representation of the database as "seen" by the DBMS, it maps the conceptual model to the DBMS. An **internal schema** depicts a specific representation of an internal model, using the database constructs supported by the chosen database.

Because this course 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 phase that matches standard language for relational databases. A simplified version of the internal model for Tiny College is shown in Figure 2.9.

FIGURE 2.9 INTERNAL MODEL FOR TINY COLLEGE



Contrary to the hierarchical and network models the relational model requires less detail in its internal model because most RDBMSs handle data access path definition transparently. Nevertheless, relational database software usually requires specifications of data storage locations, especially in a mainframe environment.

Because the internal model depends on specific database software, it is 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 type of computer on which the software is installed.

The Physical Model

The **physical model** operates at the lowest level of abstraction, describing the way data is saved on storage media such as magnetic, solid state, or optical media. 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 hardware dependent. The precision required in the physical model's definition demands that database designers have a detailed knowledge of the hardware and software used to implement the database design.

The now-dominant relational model is aimed largely at the logical level rather than at the physical level; therefore, it does not require the physical-level details common to its predecessors but may require physical-level fine-tuning for increased performance.

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. The levels of data abstraction are summarized in Table 2.4.

TABLE 2.4

LEVELS OF DATA ABSTRACTION

MODEL	DEGREE OF ABSTRACTION	FOCUS	INDEPENDENT OF
External	High	End-user views	Hardware and software
Conceptual	↕	Global view of data (database model independent)	Hardware and software
Internal	↕	Specific database model	Hardware
Physical	Low	Storage and access methods	Neither hardware nor software

Read

Database Systems: Design, Implementation, and Management, 13th Edition Carlos Coronel and Steven Morris, Chapter Two, page 61-63

Chapter Summary/Review

- In this chapter, you learned that data consists of raw facts. Information is the result of processing data to reveal its meaning.
- Data is usually stored in a database. To implement a database and to manage its contents, you need a database management system (DBMS).
- Database design defines the database structure. A well-designed database facilitates data management and generates accurate and valuable information.
- Databases evolved from manual and then computerized file systems.
- 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.
- DBMSs 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.
- Knowledge of database technologies leads to many career opportunities in the ever-expanding IT industry. There is a variety of specialization within the database arena for a wide range of skills and expertise.
- A data model is an abstraction of a complex real-world data environment. Database designers use data models to communicate with programmers and end users. The basic data-modelling components are entities, attributes, relationships, and constraints. Business rules are used to identify and define the basic modelling components within a specific real-world environment.
- The hierarchical and network data models were early models that are no longer used, but some of the concepts are found in current data models.
- 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 modelling that complements the relational model.
- The object-oriented data model (OODM) uses objects as the basic modelling structure. 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.
- Big Data technologies provide distributed, fault-tolerant, and cost-efficient support for Big Data analytics.
- Data-modelling requirements are a function of different data views (global versus local) and the level of data abstraction. There are three main levels of data abstraction: external, conceptual, and internal. The fourth and lowest level of data abstraction, called the physical level, is concerned exclusively with physical storage methods.

Review Questions

1. What is data redundancy, and which characteristics of the file system can lead to it?
2. What is structural independence, and why is it important?
3. Explain the differences among data, information, and a database.
4. What is metadata?
5. What common problems do a collection of spreadsheets created by end users share with the typical file system?
6. Discuss the importance of data models.
7. How do you translate business rules into data models?
8. Explain how the entity relationship (ER) model helped produce a more structured relational database environment.
9. What is a relationship, and what three types of relationship exist?
10. Give an example of each of the three types of relationships.

Problems

Given the following structure shown in Figure P1.1, answer the problems below:

FIGURE P1.1 THE FILE STRUCTURE FOR PROBLEMS 1–4

PROJECT_CODE	PROJECT_MANAGER	MANAGER_PHONE	MANAGER_ADDRESS	PROJECT_BID_PRICE
21-5Z	Holly B. Parker	904-338-3416	3334 Lee Rd., Gainesville, FL 37123	16833460.00
25-2D	Jane D. Grant	615-898-9909	218 Clark Blvd., Nashville, TN 36362	12500000.00
25-5A	George F. Dorts	615-227-1245	124 River Dr., Franklin, TN 29185	32512420.00
25-9T	Holly B. Parker	904-338-3416	3334 Lee Rd., Gainesville, FL 37123	21563234.00
27-4Q	George F. Dorts	615-227-1245	124 River Dr., Franklin, TN 29185	10314545.00
29-2D	Holly B. Parker	904-338-3416	3334 Lee Rd., Gainesville, FL 37123	25559999.00
31-7P	William K. Moor	904-445-2719	216 Morton Rd., Stetson, FL 30155	56850000.00

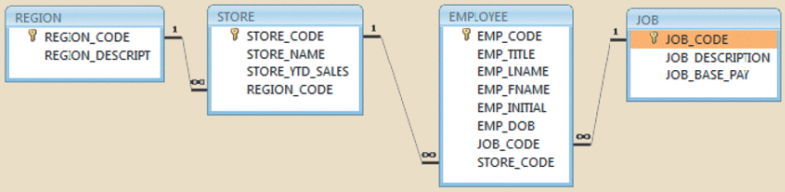
1. How many records does the file contain? How many fields are there per record?
2. What problem would you encounter if you wanted to produce a listing by city? How would you solve this problem by altering the file structure?
3. Identify and discuss the serious data redundancy problems exhibited by the file structure shown in Figure P1.9. (The file is meant to be used as a teacher class assignment schedule. One of the many problems with data redundancy is the likely occurrence of data inconsistencies—two different initials have been entered for the teacher named Maria Cordoza.)

FIGURE P1.9 THE FILE STRUCTURE FOR PROBLEMS 9–10

BUILDING_CODE	ROOM_CODE	TEACHER_LNAME	TEACHER_FNAME	TEACHER_INITIAL	DAYS_TIME
KOM	204E	Williston	Horace	G	MWF 8:00-8:50
KOM	123	Cordoza	Maria	L	MWF 8:00-8:50
LDB	504	Patroski	Donald	J	TTh 1:00-2:15
KOM	34	Hawkins	Anne	W	MWF 10:00-10:50
JKP	225B	Risell	James		TTh 9:00-10:15
LDB	301	Robertson	Jeanette	P	TTh 9:00-10:15
KOM	204E	Cordoza	Maria	I	MWF 9:00-9:50
LDB	504	Williston	Horace	G	TTh 1:00-2:15
KOM	34	Cordoza	Maria	L	MWF 11:00-11:50
LDB	504	Patroski	Donald	J	MWF 2:00-2:50

Using Figure P2.4 as your guide, work Problems 4-6. The DealCo relational diagram shows the initial entities and attributes for DealCo stores, which are in two regions of the country.

FIGURE P2.4 THE DEALCO RELATIONAL DIAGRAM



4. Identify each relationship type and write all the business rules.
5. Create the basic Crow's Foot ERD for DealCo.
6. Create the UML class diagram that reflects the entities and relationships you identified in the relational diagram.
7. United Broke Artists (UBA) is a broker for the not-so-famous artists. UBA maintains a small database to track painters, paintings, and galleries. A painting is created by a particular artist and then exhibited in a particular gallery. A gallery can exhibit many paintings, but each painting can be exhibited in only one gallery. Similarly, a painting is created by a single painter, but each painter can create many paintings. Using PAINTER, PAINTING, and GALLERY, in terms of a relational model:
    a. What tables would you create, and what would the table components be?
    b. How might the (independent) tables be related to one another?

Review Questions (MCQ)

1) What is data redundancy?

a) It is data contained in a database that is non-redundant.
b) It is data contained in a database that is accurate and consistent.
c) It is data contained in a database that is secured.
d) It is data contained in a database that is shared.

2) Which one is correct statement? Data independence provides following without changing application programs:

i) Changes in access methods.
ii) Adding new entities in database
iii) Splitting an existing record into two or more records
iv) Changing storage medium

a) i) and ii)
b) iv) only,
c) i) and iv)
d) ii) and iii)

3) Which of the following is a reason to model data?

1. Understand each user's perspective of data
2. Understand the data itself irrespective of the physical representation
3. Understand the use of data across application areas
4. All the above

4) Duplication of Data is known as?

1. Data redundancy
2. Data integrity
3. Corrupt data
4. Data dependence

5) The collection of data that contains information about an enterprise is known as?

1. DBMS
2. Database
3. Data Dictionary
4. Data file
5. Data Warehouse

6) In a relational model, relations are termed as

1. Tuples
2. Attributes
3. Tables
4. Rows

7) In a hierarchical model records are organized as

1. Graph
2. List
3. Links
4. Tree

8) The language which has recently become the de facto standard for interfacing application programs with relational database systems is

1. Oracle
2. SQL
3. DBase
4. 4GL

9) The way a particular application views the data from the database that the application uses is

1. Module
2. Relational model
3. Schema
4. Sub schema

	10) The DBMS language component which can be embedded in a program is 1. The data definition language (DDL) 2. The data manipulation language (DML) 3. The database administrator (DBA) 4. A query language

LEARNING OUTCOMES

After reading this Section of the guide, the learner should be able to:

- Give a detailed description of the relational database model's structure and apply it in database design
- Identify the relational model's basic components and explain the structure, contents and characteristics of a relational table
- Use relational database operators to manipulate relational table content
- Explain the purpose and components of the data dictionary and system catalog
- Identify appropriate entities and then the relationships among the entities in the relational database model
- Discuss how data redundancy is handled in the relational database model
- Show understanding the purpose of indexing by explanation and application in database design
- Identify the main characteristics of entity relationship components
- Give a description of how relationships between entities are defined, refined, and incorporated into the database design process
- See how entity relationship diagram components affect database design and implementation
- Describe the main extended entity relationship (EER) model constructs and how they are represented in ERDs and EERDs
- Use entity clusters to represent multiple entities and relationships in an entity relationship diagram (ERD)
- Describe characteristics of good primary keys and how to select them
- Apply flexible solutions for special data-modeling cases

A Logical View of Data

In Chapter 1.1, 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. The relational data model allows the designer to focus on the logical representation of the data and its relationships, rather than on the physical storage details and it 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 table does resemble a file from a conceptual point of view. Because the table plays such a prominent role in the relational model, it deserves a closer look. Therefore, our discussion begins by exploring the details of table structure and contents.

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 two-dimensional structure composed of rows and columns. 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 Table 3.1.

TABLE 3.1

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 intersection of a row and column 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 combination of attributes that uniquely identifies each row.

Another important characteristic, not mentioned in Table 3.1, is:

- *Each table must have a primary key*. In general terms, the primary key (PK) is an attribute or combination of attributes that uniquely identifies any given row.

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. A key consists of one or more attributes that determine other attributes. 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 super-keys, candidate keys, and secondary keys.

Dependencies

The role of a key is based on the concept of determination. **Determination** is the state in which knowing the value of one attribute makes it possible to determine the value of another. Determination in a database environment, is normally based on the relationships among the attributes.

A specific terminology and notation are used to describe relationships based on determination. The relationship is called **functional dependence**, which means that the value of one or more attributes determines the value of one or more other attributes. In this functional dependency, the attribute whose value determines another is called the determinant or the key. The attribute whose value is determined by the other attribute is called the dependent. Using this terminology, it would be correct to say that in a Student table, STU_NUM is the determinant and STU_LNAME is the dependent, since STU_NUM functionally determines STU_LNAME, and STU_LNAME is functionally dependent on STU_NUM. Also, functional dependence can involve a determinant that comprises more than one attribute and multiple dependent attributes.

Determinants made of more than one attribute require special consideration. It is possible to have a functional dependency in which the determinant contains attributes that are not necessary for the relationship. Thus, a more specific term, full functional dependence, is used to refer to functional dependencies in which the entire collection of attributes in the determinant is necessary for the relationship.

Types of Keys

Recall that a key is an attribute or group of attributes that can determine the values of other attributes. Therefore, keys are determinants in functional dependencies. Several different types of keys are used in the relational model, and you need to be familiar with them. A **composite key** is a key that is composed of more than one attribute. An attribute that is a part of a key is called a key attribute.

A **super-key** is a key that can uniquely identify any row in the table. In other words, a super-key functionally determines every attribute in the row. Furthermore, a composite that contains a super-key is also a super-key. However, it is worth noting that not all keys are super-keys.

One specific type of super-key is called a candidate key. A **candidate key** is a minimal super-key—that is, a super-key without any unnecessary attributes. A candidate key is based on a full functional dependency. A table can have many different candidate keys. Candidate keys are so called because they are the eligible options from which the designer will choose when selecting the primary key. The primary key is the candidate key chosen to be the primary means by which the rows of the table are uniquely identified. Entity integrity is the condition in which each row (entity instance) in the table has its own unique identity. To ensure entity integrity, the primary key has two requirements: (1) all of the values in the primary key must be unique and (2) no key attribute in the primary key can contain a null.

A **null** is the absence of any data value, and it is never allowed in any part of the primary key. As a rule, nulls should be avoided as much as reasonably possible. In fact, an abundance of nulls is often a sign of a poor design. Also, nulls should be avoided in the database because their meaning is not always identifiable. For example, a null could represent any of the following:

- 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.

In addition to its role in providing a unique identity to each row in the table, the primary key may play an additional role in the controlled redundancy that allows the relational model to work.

Just as the primary key has a role in ensuring the integrity of the database, so does the foreign key. Foreign keys are used to ensure referential integrity, the condition in which every reference to an entity instance by another entity instance is valid. In other words, every foreign key entry must either be null or a valid value in the primary key of the related table. Finally, a **secondary key** is defined as a key that is used strictly for data retrieval purposes, where necessary. Keep in mind that a secondary key does not necessarily yield a unique outcome.

A secondary key's effectiveness in narrowing down a search depends on how restrictive the key is. Table 3.3 summarizes the various relational database table keys.

TABLE 3.3

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
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
Secondary key	An attribute or combination of attributes used strictly for data retrieval purposes

Read

Database Systems: Design, Implementation, and Management, 13th Edition Carlos Coronel and Steven Morris, Chapter three, page 69-72

Integrity Rules

Relational database integrity rules are very important to good database design. Relational database management systems (RDBMSs) enforce integrity rules automatically, but it is much safer to make sure your application design conforms to the entity and referential integrity rules mentioned in this chapter. Those rules are summarized in Table 3.4.

The integrity rules are summarized in Table 3.4.

TABLE 3.4

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.
REFERENTIAL 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).

The next section (Chapter 2.2), Entity Relationship (ER) Modelling, discusses several ways to handle nulls. Other integrity rules that can be enforced in the relational model are the NOT NULL and UNIQUE constraints. The NOT NULL constraint can be placed on a column to ensure that every row in the table has a value for that column. The UNIQUE constraint is a restriction placed on a column to ensure that no duplicate values exist for that column.

Read

Database Systems: Design, Implementation, and Management, 13th Edition Carlos Coronel and Steven Morris, Chapter Three page 72-76

Relational Algebra

The data in relational tables is 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 relational operators. In Chapter 4, Working with Structured Query Language (SQL), you will learn how SQL commands can be used to accomplish relational algebra operations.

Formal Definitions and Terminology

As previously explained, it is common to use the terms relation and table interchangeably. However, since the mathematical terms need to be precise, we will use the more specific term relation when discussing the formal definitions of the various relational algebra operators. One important aspect of using the specific term relation is that it acknowledges the distinction between the relation and the relation variable, or *relvar*, for short. A relation is the data that we see in our tables. A ***relvar*** is a variable that holds a relation. To conveniently maintain this distinction in formulas, an unspecified

relation is often assigned a lowercase letter (e.g., "r"), while the *relvar* is assigned an uppercase letter (e.g., "R"). We could then say that r is a relation of type R, or r(R).

Relational Set Operators

The relational operators have the property of closure; that is, the use of relational algebra operators on existing relations (tables) produces new relations. Numerous operators have been defined. Some operators are fundamental, while others are convenient but can be derived using the fundamental operators.

Read

Database Systems: Design, Implementation, and Management, 13th Edition Carlos Coronel and Steven Morris, Chapter Three page 78-87

The Data Dictionary and System Catalogue

The **data dictionary** provides a detailed description of all tables in the database created by the user and designer. 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.

The **system catalogue** can be described as a detailed system data dictionary that describes all objects within the database, including data about table names, table's creator and creation date, number of columns in each table, data type corresponding to each column, index filenames, index creators, authorized users, and access privileges.

Current relational database software generally provides only a system catalogue, from which the designer's data dictionary information may be derived. The system catalogue is a system-created database whose tables store the user/designer-created database characteristics and contents. Therefore, the system catalogue tables can be queried just like any user/ designer-created table.

In effect, the system catalogue 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 a database context, the word homonym indicates the use of the same name to label different attributes. 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 whenever possible.

Read

Database Systems: Design, Implementation, and Management, 13th Edition Carlos Coronel and Steven Morris, Chapter Three page 87-89

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.

The 1:M Relationship

The 1:M relationship is the norm for relational databases. To see how such a relationship is modelled and implemented, consider the PAINTER and PAINTING example shown in Figure 3.17.

Compare the data model in Figure 3.17 with its implementation in Figure 3.18. As you examine the PAINTER and PAINTING table contents in Figure 3.18, note the following features:

- Each painting was created by one and only one painter, but each painter could have created many paintings. Note that painter 123 (Georgette P. Ross) has three works stored in the PAINTING table.
- There is only one row in the PAINTER table for any given row in the PAINTING table, but there may be many rows in the PAINTING table for any given row in the PAINTER table.

The 1:1 Relationship

As the 1:1 label implies, one entity in a 1:1 relationship can be related to only one other entity, and vice versa. The basic 1:1 relationship implementation is shown in Figure 3.22.

FIGURE 3.22 THE IMPLEMENTED 1:1 RELATIONSHIP BETWEEN PROFESSOR AND DEPARTMENT

Table name: PROFESSOR
Primary key: EMP_NUM
Foreign key: DEPT_CODE

Database name: Ch03_TinyCollege

EMP_NUM	DEPT_CODE	PROF_OFFICE	PROF_EXTENSION	PROF_HIGH_DEGREE
103	HIST	DRE 156	6783	Ph.D.
104	ENG	DRE 102	5561	MA
105	ACCT	KLR 229D	8665	Ph.D.
106	MKT/MGT	KLR 126	3899	Ph.D.
110	BIOL	AAK 160	3412	Ph.D.
114	ACCT	KLR 211	4436	Ph.D.
155	MATH	AAK 201	4440	Ph.D.
160	ENG	DRE 102	2248	Ph.D.
162	CIS	KLR 203E	2359	Ph.D.
191	MKT/MGT	KLR 409B	4016	DBA
195	PSYCH	AAK 297	3550	Ph.D.
209	CIS	KLR 333	3421	Ph.D.
228	CIS	KLR 300	3000	Ph.D.
297	MATH	AAK 194	1145	Ph.D.
299	ECON/FIN	KLR 284	2851	Ph.D.
301	ACCT	KLR 244	4683	Ph.D.
335	ENG	DRE 208	2000	Ph.D.
342	SOC	BBG 208	5514	Ph.D.
387	BIOL	AAK 230	8665	Ph.D.
401	HIST	DRE 156	6783	MA
425	ECON/FIN	KLR 284	2851	MBA
435	ART	BBG 185	2278	Ph.D.

The 1:M DEPARTMENT employs PROFESSOR relationship is implemented through the placement of the DEPT_CODE foreign key in the PROFESSOR table.

The 1:1 PROFESSOR chairs DEPARTMENT relationship is implemented through the placement of the EMP_NUM foreign key in the DEPARTMENT table.

Table name: DEPARTMENT
Primary key: DEPT_CODE
Foreign key: EMP_NUM

DEPT_CODE	DEPT_NAME	SCHOOL_CODE	EMP_NUM	DEPT_ADDRESS	DEPT_EXTENSION
ACCT	Accounting	BUS	114	KLR 211, Box 52	3119
ART	Fine Arts	A&SCI	435	BBG 185, Box 128	2278
BIOL	Biology	A&SCI	387	AAK 230, Box 415	4117
CIS	Computer Info. Systems	BUS	209	KLR 333, Box 56	3245
ECON/FIN	Economics/Finance	BUS	299	KLR 284, Box 63	3126
ENG	English	A&SCI	160	DRE 102, Box 223	1004
HIST	History	A&SCI	103	DRE 156, Box 284	1867
MATH	Mathematics	A&SCI	297	AAK 194, Box 422	4234
MKT/MGT	Marketing/Management	BUS	106	KLR 126, Box 55	3342
PSYCH	Psychology	A&SCI	195	AAK 297, Box 438	4110
SOC	Sociology	A&SCI	342	BBG 208, Box 132	2008

As you examine the tables in Figure 3.22, note several important features:

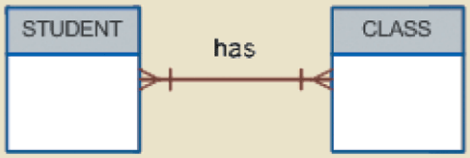
- Each professor is a Tiny College employee. Therefore, the professor identification is through the EMP_NUM. (However, note that not all employees are professors— there's another optional relationship.)
- The 1:1 "PROFESSOR chairs DEPARTMENT" relationship is implemented by having the EMP_NUM foreign key in the DEPARTMENT table.

The preceding "PROFESSOR chairs DEPARTMENT" example illustrates a proper 1:1 relationship. In fact, the use of a 1:1 relationship ensures that two entity sets are not placed in the same table when they should not be. However, the existence of a 1:1 relationship sometimes means that the entity components were not defined properly. It could indicate that the two entities actually belong in the same table! Although 1:1 relationships should be rare, certain conditions absolutely require their use.

The M:N Relationship

A many-to-many (M:N) relationship is not supported directly in the relational environment. However, M:N relationships can be implemented by creating a new entity in 1:M relationships with the original entities. To explore the M:N relationship, consider a typical college environment. The ER model in Figure 3.23 shows this M:N relationship.

FIGURE 3.23 THE ERM'S M:N RELATIONSHIP BETWEEN STUDENT AND CLASS



Note the features of the ERM in Figure 3.23.

- Each CLASS can have many STUDENTs, and each STUDENT can take many CLASSes.
- There can be many rows in the CLASS table for any given row in the STUDENT table, and there can be many rows in the STUDENT table for any given row in the CLASS table.

However, M:N relationships should not be implemented for two good reasons:

- The tables create many redundancies which lead to anomalies.
- The relational operations become very complex and are likely to lead to system efficiency errors and output errors.

Fortunately, the problems inherent in the M:N relationship can easily be avoided by creating a composite entity (also referred to as a bridge entity or an associative entity). Because such a table is used to link the tables that were originally related in an M:N relationship, the composite entity structure includes—as foreign keys—at least the primary keys of the tables that are to be linked. The database designer has two main options when defining a composite table's primary key: use the combination of those foreign keys or create a new primary key.

Remember that each entity in the ERM is represented by a table. Therefore, you can create the composite ENROLL table shown in Figure 3.25 to link the tables CLASS and STUDENT.

FIGURE 3.25 CONVERTING THE M:N RELATIONSHIP INTO TWO 1:M RELATIONSHIPS

Table name: STUDENT
Primary key: STU_NUM
Foreign key: none

Database name: Ch03_CollegeTry2

STU_NUM	STU_LNAME
321452	Bowser
324257	Smithson

Table name: ENROLL
Primary key: CLASS_CODE + STU_NUM
Foreign key: CLASS_CODE, STU_NUM

CLASS_CODE	STU_NUM	ENROLL_GRADE
10014	321452	C
10014	324257	B
10018	321452	A
10018	324257	B
10021	321452	C
10021	324257	C

Table name: CLASS
Primary key: CLASS_CODE
Foreign key: CRS_CODE

CLASS_CODE	CRS_CODE	CLASS_SECTION	CLASS_TIME	CLASS_ROOM	PROF_NUM
10014	ACCT-211	3	TTh 2:30-3:45 p.m.	BUS252	342
10018	CIS-220	2	MWF 9:00-9:50 a.m.	KLR211	114
10021	QM-261	1	MWF 8:00-8:50 a.m.	KLR200	114

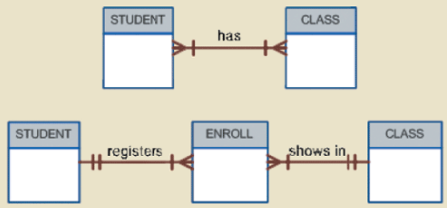
Because the ENROLL table in Figure 3.25 links two tables, STUDENT and CLASS, it is also called a linking table. In other words, a linking table is the implementation of a composite entity.

The ENROLL table shown in Figure 3.25 yields the required M:N to 1:M conversion. Observe that the composite entity represented by the ENROLL table must contain at least the primary keys of the CLASS and STUDENT tables (CLASS_CODE and STU_ NUM, respectively) for which it

serves as a connector. Also note that the STUDENT and CLASS tables now contain only one row per entity.

Naturally, the conversion is reflected in the ERM, too. The revised relationship is shown in Figure 3.26. As you examine Figure 3.26, note that the composite entity named ENROLL represents the linking table between STUDENT and CLASS.

FIGURE 3.26 CHANGING THE M:N RELATIONSHIPS TO TWO 1:M RELATIONSHIPS



Read

Database Systems: Design, Implementation, and Management, 13th Edition Carlos Coronel and Steven Morris, Chapter Three page 89-97

Data Redundancy Revisited

In Chapter 1, you learned that data redundancy leads to data anomalies, which 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 they do not totally eliminate the problem because the foreign key values can be repeated many times. However, the proper use of foreign keys minimizes data redundancies and the chances that destructive data anomalies will develop.

You will learn in Chapter 2.2 that database designers must reconcile three often contradictory requirements: design elegance, processing speed, and information requirements. Also, you will learn that proper data warehousing design requires carefully defined and controlled data redundancies to function properly. Regardless of how you describe data redundancies, the potential for damage is limited by proper implementation and careful control. As important as it is to control data redundancy, sometimes the level of data redundancy must be increased to make the database serve crucial information purposes. Also, data redundancies sometimes seem to exist to preserve the historical accuracy of the data. Thus, such planned “redundancies” are common in good database design.

Read

Database Systems: Design, Implementation, and Management, 13th Edition Carlos Coronel and Steven Morris, Chapter Three page 97-99

Indexes

Suppose you want to locate a book in a library. Does it make sense to look through every book until you find the one you want? Of course not; you use the library's catalogue, which is indexed by title, topic, and author. The index (in either a manual or computer library catalogue) points you to the book's location, making retrieval a quick and simple matter. An **index** is an orderly arrangement used to logically access rows in a table more efficiently.

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.

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. Also, an index key can be composed of one or more attributes. 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.

In a unique index, as its name implies, the index key can have only one pointer value (row) associated with it. A table can have many indexes, but each index is associated with only one table. The index key can have multiple attributes (a composite index).

Read

Database Systems: Design, Implementation, and Management, 13th Edition Carlos Coronel and Steven Morris, Chapter Three page 99-100

The Entity Relationship (ER) Model

Recall from Chapter 2, Data Models, and Chapter 3, The Relational Database Model, that the entity relationship model (ERM) forms the basis of an ERD. The ERD represents the conceptual database as viewed by the end user. ERDs depict the database's main components: entities, attributes, and relationships. Because an entity represents a real-world object, the words entity and object are often used interchangeably.

Earlier, you also learned about the various notations used with ERDs—the original Chen notation and the newer Crow's Foot and UML notations. Some conceptual database modelling concepts can be expressed only using the Chen notation. Because of its emphasis on implementation, the Crow's Foot notation can represent only what could be implemented. In other words:

- The Chen notation favours conceptual modelling.
- The Crow's Foot notation favours a more implementation-oriented approach.
- The UML notation can be used for both conceptual and implementation modelling.

Entities

An entity in the ERM corresponds to a table—not to a row—in the relational environment. The ERM refers to a table row as an entity instance or entity occurrence. In the Chen, Crow's Foot, and UML notations, an entity is represented by a rectangle that contains the entity's name. The entity name, a noun, is usually written in all capital letters.

Attributes

Attributes are characteristics of entities. In the Crow's Foot notation, the attributes are written in the attribute box below the entity rectangle. (See Figure 4.1.)

Required and Optional Attributes A required attribute is an attribute that must have a value; in other words, it cannot be left empty. As shown in Figure 4.1, the two boldfaced attributes in the Crow's Foot notation indicate that data entry will be required. An optional attribute is an attribute that does not require a value; therefore, it can be left empty.

Domains Attributes have a domain. A domain is the set of possible values for a given attribute. Attributes may share a domain. In fact, the data dictionary may let a newly declared attribute inherit the characteristics of an existing attribute if the same attribute name is used.

Identifiers (Primary Keys) The ERM uses identifiers—one or more attributes that uniquely identify each entity instance. In the relational model, entities are mapped to tables, and the entity identifier is mapped as the table's primary key (PK). Identifiers are underlined in the ERD. Key attributes are also underlined in a frequently used shorthand notation for the table structure, called a relational schema.

Composite Identifiers Ideally, an entity identifier is composed of only a single attribute. However, it is possible to use a composite identifier, a primary key composed of more than one attribute.

Composite and Simple Attributes. Attributes are classified as simple or composite. A composite attribute, not to be confused with a composite key, is an attribute that can be further subdivided to yield additional attributes. A simple attribute is an attribute that cannot be subdivided. To facilitate detailed queries, it is wise to change composite attributes into a series of simple attributes.

Single-Valued Attributes A single-valued attribute is an attribute that can have only a single value. For example, a person can have only one ID number, and a manufactured part can have only one serial number. Keep in mind that a single-valued attribute is not necessarily a simple attribute.

Multivalued Attributes Multivalued attributes are attributes that can have many values. For instance, a person may have several college degrees. In the Chen ERM, multivalued attributes are shown by a double line connecting the attribute to the entity. The Crow's Foot notation does not identify multivalued attributes.

Implementing Multivalued Attributes Although the conceptual model can handle M:N relationships and multivalued attributes, you should not implement them in the RDBMS. Remember that in the relational table, each column and row intersection represents a single data value. So, if multivalued attributes exist, the designer must decide on one of two possible actions:

1. Within the original entity, create several new attributes, one for each component of the original multivalued attribute. Although this solution seems to work, its adoption can lead to major structural problems in the table and it is not always acceptable.
2. Create a new entity composed of the original multivalued attribute's components. This is the preferred way to deal with multivalued attributes. Creating a new entity in a 1:M relationship with the original entity yields several benefits: it is a more flexible, expandable solution, and it is compatible with the relational model!

Derived Attributes Finally, a derived attribute is an attribute whose value is calculated (derived) from other attributes. The derived attribute need not be physically stored within the database; instead, it can be derived by using an algorithm. A derived attribute is indicated in the Chen notation by a dashed line that connects the attribute and the entity. (See Figure 4.6.) The Crow's Foot notation does not have a method for distinguishing the derived attribute from other attributes. Derived attributes are sometimes referred to as computed attributes. The decision to store derived attributes in database tables depends on the processing requirements and the constraints placed on an application. The designer should be able to balance the design in accordance with such constraints. Table 4.2 shows the advantages and disadvantages of storing (or not storing) derived attributes in the database.

Relationships

Remember that a relationship is an association between entities. The entities that participate in a relationship are also known as participants, and each relationship is identified by a name that describes the relationship. The relationship name is an active or passive verb; for example, a STUDENT takes a CLASS, a PROFESSOR teaches a CLASS, a DEPARTMENT employs a PROFESSOR, a DIVISION is managed by an EMPLOYEE, and an AIRCRAFT is flown by a CREW.

Relationships between entities always operate in both directions. To define the relationship between the entities named CUSTOMER and INVOICE, you would specify that:

- A CUSTOMER may generate many INVOICEs.
- Each INVOICE is generated by one CUSTOMER.

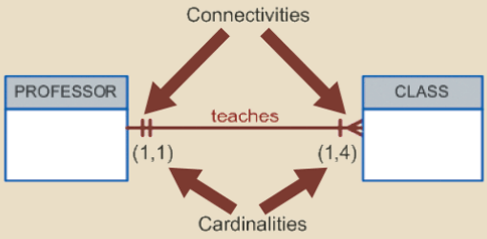
Because you know both directions of the relationship between CUSTOMER and INVOICE, it is easy to see that this relationship can be classified as 1:M.

Connectivity and Cardinality

You learned in Chapter 1.2 that entity relationships may be classified as one-to-one, one- to-many, or many-to-many. You also learned how such relationships were depicted in the Chen and Crow's Foot notations. The term **connectivity** is used to describe the relationship classification.

Cardinality expresses the minimum and maximum number of entity occurrences associated with one occurrence of the related entity. In the ERD, cardinality is indicated by placing the appropriate numbers beside the entities, using the format (x, y). The first value represents the minimum number of associated entities, while the second value represents the maximum number of associated entities. When the specific cardinalities are not included on the diagram in Crow's Foot notation, cardinality is implied by the symbols shown in Figure 4.7, which describe the connectivity and participation (discussed next).

FIGURE 4.7 CONNECTIVITY AND CARDINALITY IN AN ERD



Knowing the minimum and maximum number of entity occurrences is very useful at the application software level. However, keep in mind that the DBMS cannot handle the implementation of the cardinalities at the table level—that capability is provided by the application software or by triggers.

As you examine the Crow's Foot diagram in Figure 4.7, keep in mind that the cardinalities represent the number of occurrences in the related entity. For example, the cardinality (1,4) next to the CLASS entity in the "PROFESSOR teaches CLASS" relationship indicates that each professor teaches up to four classes, which means that the PROFESSOR table's primary key value occurs at least once and no more than four times as foreign key values in the CLASS table.

Connectivities and cardinalities are established by concise statements known as business rules, which were introduced in Chapter 1.2. Such rules, derived from a precise and detailed description of an organization's data environment, also establish the ERM's entities, attributes, relationships, connectivities, cardinalities, and constraints. Because business rules define the ERM's components, making sure that all appropriate business rules are identified is an important part of a database designer's job.

Existence Dependence

An entity is said to be **existence-dependent** if it can exist in the database only when it is associated with another related entity occurrence. In implementation terms, an entity is existence-dependent if it has a mandatory foreign key—that is, a foreign key attribute that cannot be null.

If an entity can exist apart from its related entities, then it is **existence-independent**, and it is referred to as a **strong entity** or **regular entity**.

Relationship Strength

The concept of relationship strength is based on how the primary key of a related entity is defined. To implement a relationship, the primary key of one entity (the parent entity, normally on the "one" side of the one-to-many relationship) appears as a foreign key in the related entity (the child entity, mostly the entity on the "many" side of the one- to-many relationship). Sometimes, the foreign key

also is a primary key component in the related entity. In this section, you will learn how various relationship strength decisions affect primary key arrangement in database design.

Weak (Non-Identifying) Relationships A weak relationship, also known as a non-identifying relationship, exists if the primary key of the related entity does not contain a primary key component of the parent entity. By default, relationships are established by having the primary key of the parent entity appear as a foreign key (FK) on the related entity (also known as the child entity).

Figure 4.8 shows how the Crow's Foot notation depicts a weak relationship by placing a dashed relationship line between the entities. The tables shown below the ERD illustrate how such a relationship is implemented.

Strong (Identifying) Relationships A strong (identifying) relationship exists when the primary key of the related entity contains a primary key component of the parent entity. The Crow's Foot notation depicts the strong (identifying) relationship with a solid line between the entities.

In summary, whether the relationship between COURSE and CLASS is strong or weak depends on how the CLASS entity's primary key is defined. Remember that the nature of the relationship is often determined by the database designer, who must use professional judgment to determine which relationship type and strength best suit the database transaction, efficiency, and information requirements.

Weak Entities

In contrast to the strong or regular entity mentioned earlier, a weak entity is one that meets two conditions:

1. The entity is existence-dependent; it cannot exist without the entity with which it has a relationship.
2. The entity has a primary key that is partially or totally derived from the parent entity in the relationship.

The Crow's Foot notation uses the relationship line and the PK/FK designation to indicate whether the related entity is weak. A strong (identifying) relationship indicates that the related entity is weak. Such a relationship means that both conditions have been met for the weak entity definition—the related entity is existence-dependent, and the PK of the related entity contains a PK component of the parent entity. Remember that the weak entity inherits part of its primary key from its strong counterpart.

Keep in mind that the database designer usually determines whether an entity can be described as weak based on the business rules. An examination of Figure 4.8 might cause you to conclude that CLASS is a weak entity to COURSE. After all, it seems clear that a CLASS cannot exist without a COURSE, so there is existence dependence. The second weak entity requirement has not been met; therefore, by definition, the CLASS entity in Figure 4.8 may not be classified as weak.

FIGURE 4.8 A WEAK (NON-IDENTIFYING) RELATIONSHIP BETWEEN COURSE AND CLASS

Table name: COURSE **Database name: Ch04_TinyCollege**

CRS_CODE	DEPT_CODE	CRS_DESCRIPTION	CRS_CREDIT
ACCT-211	ACCT	Accounting I	3
ACCT-212	ACCT	Accounting II	3
CIS-220	CIS	Intro. to Microcomputing	3
CIS-420	CIS	Database Design and Implementation	4
MATH-243	MATH	Mathematics for Managers	3
QM-261	CIS	Intro. to Statistics	3
QM-362	CIS	Statistical Applications	4

Table name: CLASS

CLASS_CODE	CRS_CODE	CLASS_SECTION	CLASS_TIME	ROOM_CODE	PROF_NUM
10012	ACCT-211	1	MWF 8:00-8:50 a.m.	BUS311	105
10013	ACCT-211	2	MWF 9:00-9:50 a.m.	BUS200	105
10014	ACCT-211	3	TTh 2:30-3:45 p.m.	BUS252	342
10015	ACCT-212	1	MWF 10:00-10:50 a.m.	BUS311	301
10016	ACCT-212	2	Th 6:00-8:40 p.m.	BUS252	301
10017	CIS-220	1	MWF 9:00-9:50 a.m.	KLR209	228
10018	CIS-220	2	MWF 9:00-9:50 a.m.	KLR211	114
10019	CIS-220	3	MWF 10:00-10:50 a.m.	KLR209	228
10020	CIS-420	1	W 6:00-8:40 p.m.	KLR209	162
10021	QM-261	1	MWF 8:00-8:50 a.m.	KLR200	114
10022	QM-261	2	TTh 1:00-2:15 p.m.	KLR200	114
10023	QM-362	1	MWF 11:00-11:50 a.m.	KLR200	162
10024	QM-362	2	TTh 2:30-3:45 p.m.	KLR200	162
10025	MATH-243	1	Th 6:00-8:40 p.m.	DRE155	325

On the other hand, if the CLASS entity's primary key had been defined as a composite key composed of the combination CRS_CODE and CLASS_SECTION, CLASS could be represented by: CLASS (CRS_CODE, CLASS_SECTION, CLASS_TIME, ROOM_CODE, PROF_NUM) In that case, the CLASS primary key is partially derived from COURSE because CRS_CODE is the COURSE table's primary key. Given this decision, CLASS is a weak entity. In any case, CLASS is always existence-dependent on COURSE, whether or not it is defined as weak.

Relationship Participation

Participation in an entity relationship is either optional or mandatory. Recall that relationships are bidirectional; that is, they operate in both directions. If COURSE is related to CLASS, then by definition, CLASS is related to COURSE. Because of the bidirectional nature of relationships, it is necessary to determine the connectivity of the relationship from both directions. Similarly, the specific maximum and minimum cardinalities must be determined in each direction for the relationship. Once again, you must consider the bidirectional nature of the relationship when determining participation.

Optional participation means that one entity occurrence does not require a corresponding entity occurrence in a relationship. For example, in the "COURSE generates CLASS" relationship, you noted that at least some courses do not generate a class. In other words, an entity occurrence (row) in the COURSE table does not necessarily require the existence of a corresponding entity

occurrence in the CLASS table. Therefore, the CLASS entity is considered optional to the COURSE entity. In Crow's Foot notation, an optional relationship between entities is shown by drawing a small circle (O) on the side of the optional entity, as illustrated in Figure 4.9. The existence of an optional entity indicates that its minimum cardinality is 0.

Mandatory participation means that one entity occurrence requires a corresponding entity occurrence in a relationship. If no optionality symbol is depicted with the entity, the entity is assumed to exist in a mandatory relationship with the related entity. If the mandatory participation is depicted graphically, it is typically shown as a small hash mark across the relationship line, similar to the Crow's Foot depiction of a connectivity of 1. The existence of a mandatory relationship indicates that the minimum cardinality is at least 1 for the mandatory entity.

Because relationship participation is an important component of database design, it is important that you clearly understand the distinction between mandatory and optional participation in relationships. Otherwise, you might develop designs in which awkward and unnecessary temporary rows (entity instances) must be created just to accommodate the creation of required entities. It is also important to understand that the semantics of a problem might determine the type of participation in a relationship.

Relationship Degree

A relationship degree indicates the number of entities or participants associated with a relationship. A **unary relationship** exists when an association is maintained within a single entity. A **binary relationship** exists when two entities are associated. A **ternary relationship** exists when three entities are associated. Although higher degrees exist, they are rare and are not specifically named. (For example, an association of four entities is described simply as a four-degree relationship.) Figure 4.15 shows these types of relationship degrees.

Recursive Relationships

As you just learned, a **recursive relationship** is one in which a relationship can exist between occurrences of the same entity set. (Naturally, such a condition is found within a unary relationship.) For example, a 1:M unary relationship can be expressed by "an EMPLOYEE may manage many EMPLOYEEs, and each EMPLOYEE is managed by one EMPLOYEE."

The M:N recursive relationship might be more familiar in a school environment. For instance, note how the M:N "COURSE requires COURSE" relationship is implemented in Figure 4.21. In this example, MATH-243 is a prerequisite to QM-261 and QM-362, while both MATH-243 and QM-261 are prerequisites to QM-362.

FIGURE 4.21 IMPLEMENTATION OF THE M:N RECURSIVE RELATIONSHIP "COURSE REQUIRES COURSE"

Table name: COURSE **Database name: Ch04_TinyCollege**

CRS_CODE	DEPT_CODE	CRS_DESCRIPTION	CRS_CREDIT
ACCT-211	ACCT	Accounting I	3
ACCT-212	ACCT	Accounting II	3
CIS-220	CIS	Intro. to Microcomputing	3
CIS-420	CIS	Database Design and Implementation	4
MATH-243	MATH	Mathematics for Managers	3
QM-261	CIS	Intro. to Statistics	3
QM-362	CIS	Statistical Applications	4

Table name: PREREQ

CRS_CODE	PRE_TAKE
CIS-420	CIS-220
QM-261	MATH-243
QM-362	MATH-243
QM-362	QM-261

One common pitfall when working with unary relationships is to confuse participation with referential integrity. In theory, participation and referential integrity are very different concepts and are normally easy to distinguish in binary relationships. In practical terms, conversely, participation and referential integrity are very similar because they are both implemented through constraints on the same set of attributes. This similarity often leads to confusion when the concepts are applied within the limited structure of a unary relationship.

Referential integrity requires that the values in the foreign key correspond to values in the primary key. In one direction, participation considers whether the foreign key can contain a null.

Associative (Composite) Entities

M:N relationships are a valid construct at the conceptual level, and therefore are found frequently during the ER modelling process. However, implementing the M:N relationship, particularly in the relational model, requires the use of an additional entity. The ER model uses the associative entity to represent an M:N relationship between two or more entities. This **associative entity**, also called a composite or bridge entity, is in a 1:M relationship with the parent entities and is composed of the primary key attributes of each parent entity. Furthermore, the associative entity can have additional attributes of its own, as shown by the ENROLL associative entity in Figure 4.23. When using the Crow's Foot notation, the associative entity is identified as a strong (identifying) relationship, as indicated by the solid relationship lines between the parents and the associative entity.

FIGURE 4.23 CONVERTING THE M:N RELATIONSHIP INTO TWO 1:M RELATIONSHIPS

Table name: STUDENT — **Database name: Ch04_CollegeTry**

STU_NUM	STU_LNAME
321452	Bowser
324257	Smithson

Table name: ENROLL

CLASS_CODE	STU_NUM	ENROLL_GRADE
10014	321452	C
10014	324257	B
10018	321452	A
10018	324257	B
10021	321452	C
10021	324257	C

Table name: CLASS

CLASS_CODE	CRS_CODE	CLASS_SECTION	CLASS_TIME	ROOM_CODE	PROF_NUM
10014	ACCT-211	3	TTh 2:30-3:45 p.m.	BUS252	342
10018	CIS-220	2	MWF 9:00-9:50 a.m.	KLR211	114
10021	QM-261	1	MWF 8:00-8:50 a.m.	KLR200	114

Note that the composite ENROLL entity in Figure 4.23 is existence-dependent on the other two entities; the composition of the ENROLL entity is based on the primary keys of the entities that are connected by the composite entity.

Implementing the small database shown in Figure 4.23 requires that you define the relationships clearly. Specifically, you must know the "1" and the "M" sides of each relationship, and you must know whether the relationships are mandatory or optional.

Read

Database Systems: Design, Implementation, and Management, 13th Edition Carlos Coronel and Steven Morris, Chapter Four page 114-138

Developing an ER Diagram

The process of database design is iterative rather than a linear or sequential process. The verb iterate means "to do again or repeatedly." Thus, an iterative process is based on repetition of processes and procedures. Building an ERD usually involves the following activities:

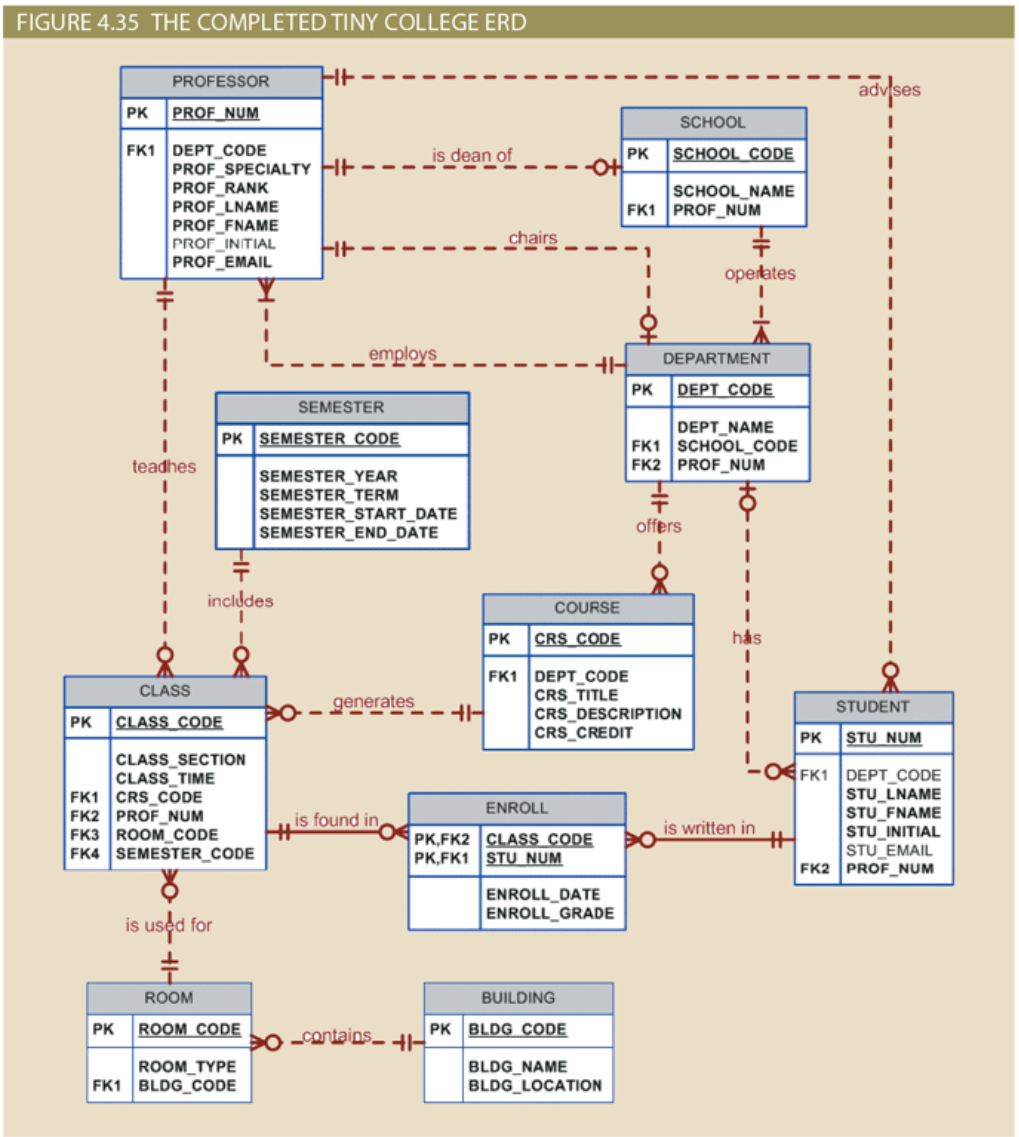
- Create a detailed narrative of the organization's description of operations.
- Identify the business rules based on the description of operations.
- Identify the main entities and relationships from the business rules.
- Develop the initial ERD.
- Identify the attributes and primary keys that adequately describe the entities.
- Revise and review the ERD. During the review process, additional objects, attributes, and relationships probably will be uncovered.

The process is repeated until the end users and designers agree that the ERD is a fair representation of the organization's activities and functions. During the design process, the database designer does not depend simply on interviews to help define entities, attributes, and relationships. A surprising amount of information can be gathered by examining the business forms and reports that an organization uses in its daily operations.

Once you have discovered the relevant entities, you can define the initial set of relationships among them. Next, you describe the entity attributes. Identifying the attributes of the entities helps you to better understand the relationships among entities. Table 4.4 summarizes the ERM's components, and names the entities and their relations.

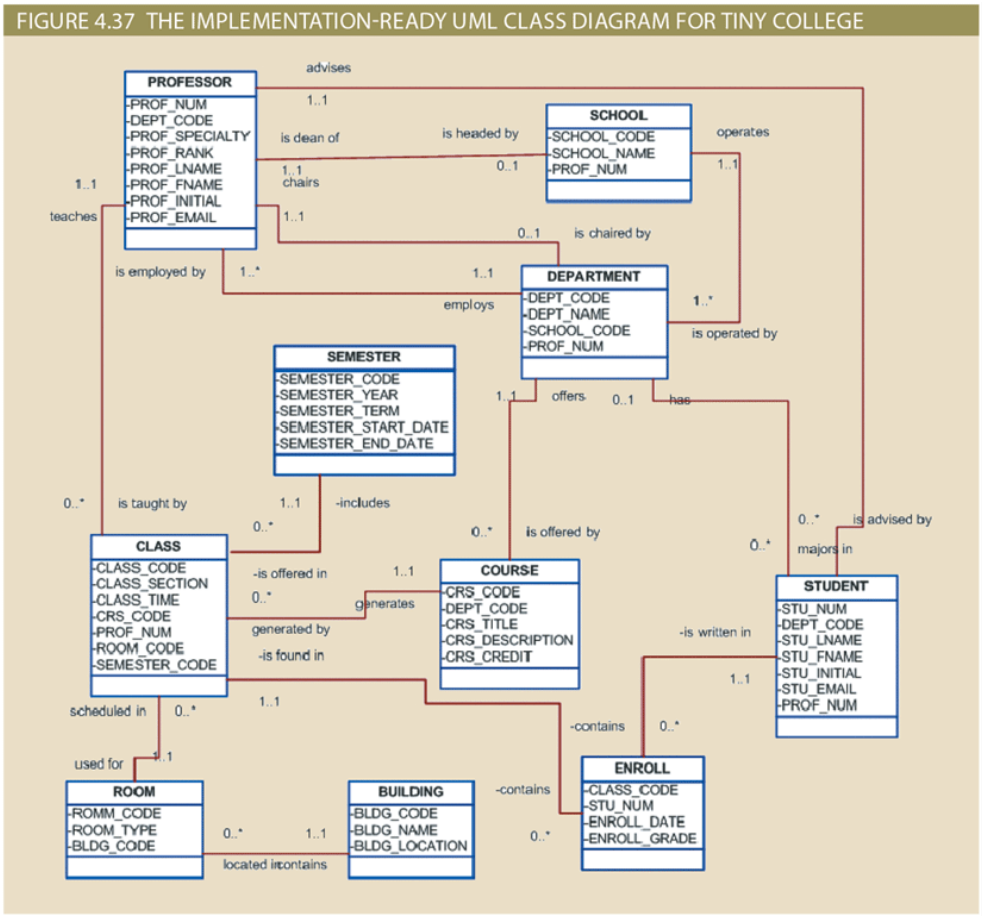
You must also define the connectivity and cardinality for the just-discovered relations based on the business rules. However, to avoid crowding the diagram, the cardinalities are not shown. Figure 4.35 shows the Crow's Foot ERD for Tiny College. Note that this is an implementation-ready model, so it shows the ENROLL composite entity.

FIGURE 4.35 THE COMPLETED TINY COLLEGE ERD



Although we focus on Crow's Foot notation to develop our diagram, UML notation is also popular for conceptual and implementation modelling. Figure 4.37 shows the implementation-ready UML class diagram for Tiny College (note that the ENROLL composite entity is shown in this class diagram).

FIGURE 4.37 THE IMPLEMENTATION-READY UML CLASS DIAGRAM FOR TINY COLLEGE



Read

Database Systems: Design, Implementation, and Management, 13th Edition Carlos Coronel and Steven Morris, Chapter Four page 138-146.

Database Design Challenges: Conflicting Goals

Database designers must often make design compromises that are triggered by conflicting goals, such as adherence to:

- *Design standards*. The database design must conform to design standards. Such standards guide you in developing logical structures that minimize data redundancies, thereby minimizing the likelihood that destructive data anomalies will occur. Without design standards, it is nearly impossible to formulate a proper design process, to evaluate an existing design, or to trace the likely logical impact of changes in design.

- *Processing speed*. High processing speed means minimal access time, which may be achieved by minimizing the number and complexity of logically desirable relationships.
- *Information requirements.* The quest for timely information might be the focus of database design. Complex information requirements may dictate data transformations, and they may expand the number of entities and attributes within the design. Therefore, the database may have to sacrifice some of its "clean" design structures and high transaction speed to ensure maximum information generation.

A design that meets all logical requirements and design conventions is an important goal. However, if this perfect design fails to meet the customer's transaction speed and information requirements, the designer will not have done a proper job from the end user's point of view.

Compromises are a fact of life in the real world of database design. Even while focusing on the entities, attributes, relationships, and constraints, the designer should begin thinking about end-user requirements such as performance, security, shared access, and data integrity. The designer must consider processing requirements and verify that all update, retrieval, and deletion options are available. Finally, a design is of little value unless the end product can deliver all specified query and reporting requirements. You will probably discover that even the best design process produces an ERD that requires further changes mandated by operational requirements.

Using ERDs yields perhaps the richest bonus of all: a thorough understanding of how an organization really functions. Occasionally, design and implementation problems do not yield "clean" implementation solutions.

Your job as a database designer is to use your professional judgment to yield a solution that meets the requirements imposed by business rules, processing requirements, and basic design principles.

Read

Database Systems: Design, Implementation, and Management, 13th Edition Carlos Coronel and Steven Morris, Chapter Five page 159-168.

The Extended Entity Relationship Model

As the complexity of the data structures being modelled has increased and as application software requirements have become more stringent, the need to capture more information in the data model has increased. **The extended entity relationship model (EERM)**, sometimes referred to as the enhanced entity relationship model, is the result of adding more semantic constructs to the original ER model. As you might expect, a diagram that uses the EERM is called an EER diagram (EERD). In the following sections, you will learn about the main EER model constructs—entity super-types, entity subtypes, and entity clustering—and see how they are represented in ERDs or EERDs.

Entity Super-types and Subtypes

In modelling terms, an **entity supertype** is a generic entity type that is related to one or more **entity subtypes**. The entity supertype contains common characteristics, and the entity subtypes each contain their own unique characteristics.

Two criteria help the designer determine when to use subtypes and supertypes:

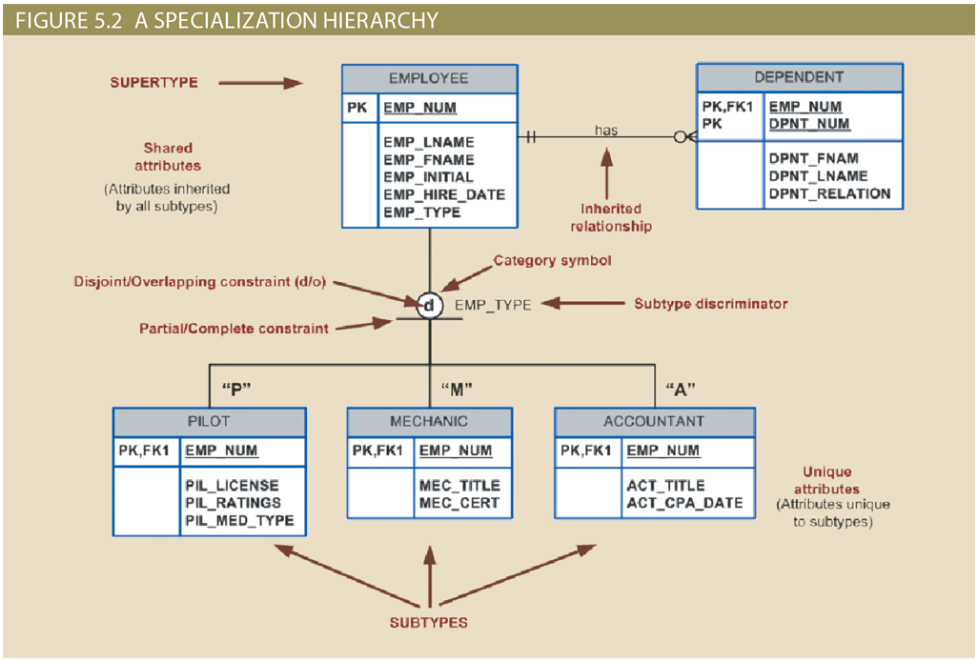
- There must be different, identifiable kinds or types of the entity in the user's environment.
- The different kinds or types of instances should each have one or more attributes that are unique to that kind or type of instance.

For example, because pilots meet both criteria of being an identifiable kind of employee and having unique attributes that other employees do not possess, it is appropriate to create PILOT as a subtype of EMPLOYEE. Assume that mechanics and accountants also each have attributes that are unique to them, respectively, and that clerks do not. In that case, MECHANIC and ACCOUNTANT would also be legitimate subtypes of EMPLOYEE because they are identifiable kinds of employees and have unique attributes. CLERK would not be an acceptable subtype of EMPLOYEE because it only satisfies one of the criteria—it is an identifiable kind of employee—but none of the attributes are unique to just clerks. In the next section, you will learn how entity super-types and subtypes are related in a specialization hierarchy.

Specialization Hierarchy

Entity supertypes and subtypes are organized in a specialization hierarchy, which depicts the arrangement of higher-level entity supertypes (parent entities) and lower-level entity subtypes (child entities). Figure 5.2 shows the specialization hierarchy formed by an EMPLOYEE supertype and three entity subtypes—PILOT, MECHANIC, and ACCOUNTANT. The specialization hierarchy reflects the 1:1 relationship between EMPLOYEE and its subtypes. For example, a PILOT subtype occurrence is related to one instance of the EMPLOYEE supertype.

FIGURE 5.2 A SPECIALIZATION HIERARCHY



The terminology and symbols in Figure 5.2 are explained throughout this chapter. The relationships depicted within the specialization hierarchy are sometimes described in terms of "is-a" relationships. For example, a pilot is an employee. It is important to understand that within a specialization hierarchy, a subtype can exist only within the context of a supertype, and every subtype can have only one supertype to which it is directly related. However, a specialization hierarchy can have many levels of supertype or subtype relationships—that is, you can have a specialization hierarchy in which a supertype has many subtypes. In turn, one of the subtypes is the supertype to other lower-level subtypes.

As you can see in Figure 5.2, the arrangement of entity supertypes and subtypes in a specialization hierarchy is more than a cosmetic convenience. Specialization hierarchies enable the data model to capture additional semantic content (meaning) into the ERD. A specialization hierarchy provides the means to:

- Support attribute inheritance.
- Define a special supertype attribute known as the subtype discriminator.
- Define disjoint or overlapping constraints and complete or partial constraints.

The following sections cover such characteristics and constraints in more detail.

Inheritance

The property of **inheritance** enables an entity subtype to inherit the attributes and relationships of the supertype. For example, Figure 5.2 illustrates that pilots, mechanics, and accountants all inherit the employee number, last name, first name, middle initial, and hire date from the EMPLOYEE entity. One important inheritance characteristic is that all entity subtypes inherit their primary key attribute from their supertype. At the implementation level, the supertype and its subtype(s) depicted in the specialization hierarchy maintain a 1:1 relationship.

Entity subtypes inherit all relationships in which the supertype entity participates. Through inheritance, all subtypes also participate in that relationship. In specialization hierarchies with multiple levels of supertype and subtypes, a lower-level subtype inherits all of the attributes and relationships from all of its upper-level supertypes. Inheriting the relationships of their supertypes does not mean that subtypes cannot have relationships of their own.

Subtype Discriminator

A subtype discriminator is the attribute in the supertype entity that determines to which subtype the supertype occurrence is related. In Figure 5.2, the subtype discriminator is the employee type (EMP_TYPE). It is common practice to show the subtype discriminator and its value for each subtype in the ERD, as shown in Figure 5.2.

Using Figure 5.2 as your guide, note that the supertype is related to a PILOT subtype if the EMP_TYPE has a value of "P." If the EMP_TYPE value is "M," the supertype is related to a MECHANIC subtype. If the EMP_TYPE value is "A," the supertype is related to the ACCOUNTANT subtype. Note that the default comparison condition for the subtype discriminator attribute is the equality comparison. However, in some situations the subtype discriminator is not necessarily based on an equality comparison.

As you learned earlier in this section, the implementation of disjoint subtypes is based on the value of the subtype discriminator attribute in the supertype. However, implementing overlapping subtypes requires the use of one discriminator attribute for each subtype. For example, in the case of the Tiny College database design, a professor can also be an administrator. Therefore, the EMPLOYEE supertype would have the subtype discriminator attributes and values shown in Table 5.1.

TABLE 5.1

DISCRIMINATOR ATTRIBUTES WITH OVERLAPPING SUBTYPES

DISCRIMINATOR ATTRIBUTES		COMMENT
PROFESSOR	ADMINISTRATOR	
Y	N	The Employee is a member of the Professor subtype.
N	Y	The Employee is a member of the Administrator subtype.
Y	Y	The Employee is both a Professor and an Administrator.

Completeness Constraint

The **completeness constraint** specifies whether each entity supertype occurrence must also be a member of at least one subtype. The completeness constraint can be partial or total. **Partial completeness** means that not every supertype occurrence is a member of a subtype; some supertype occurrences may not be members of any subtype. **Total completeness** means that every supertype occurrence must be a member of at least one subtype.

Given the disjoint and overlapping subtypes and completeness constraints, it is possible to have the specialization hierarchy constraint scenarios shown in Table 5.2.

TABLE 5.2

SPECIALIZATION HIERARCHY CONSTRAINT SCENARIOS

TYPE	DISJOINT CONSTRAINT	OVERLAPPING CONSTRAINT
Partial 	Supertype has optional subtypes. Subtype discriminator can be null. Subtype sets are unique.	Supertype has optional subtypes. Subtype discriminators can be null. Subtype sets are not unique.
Total 	Every supertype occurrence is a member of only one subtype. Subtype discriminator cannot be null. Subtype sets are unique.	Every supertype occurrence is a member of at least one subtype. Subtype discriminators cannot be null. Subtype sets are not unique.

Specialization and Generalization

You can use various approaches to develop entity supertypes and subtypes. For example, you can first identify a regular entity and then identify all entity subtypes based on their distinguishing

characteristics. You can also start by identifying multiple entity types and then later extract the common characteristics of those entities to create a higher-level supertype entity. **Specialization** is the top-down process of identifying lower-level, more specific entity subtypes from a higher-level entity supertype. Specialization is based on grouping the unique characteristics and relationships of the subtypes

Generalization is the bottom-up process of identifying a higher-level, more generic entity supertype from lower-level entity subtypes. Generalization is based on grouping the common characteristics and relationships of the subtypes.

Read

Database Systems: Design, Implementation, and Management, 13th Edition Carlos Coronel and Steven Morris, Chapter Five page 168-175.

Entity Clustering

In the case that in the process of developing an ER diagram, hundreds of entities and relationships that overcrowd the diagram are discovered, you can use entity clusters to minimize the number of entities shown in the ERD. An **entity cluster** is a "virtual" entity type used to represent multiple entities and relationships in the ERD. An entity cluster is formed by combining multiple interrelated entities into a single, abstract entity object. Instead, it is a temporary entity used to represent multiple entities and relationships, with the purpose of simplifying the ERD and thus enhancing its readability. Figure 5.6 illustrates the use of entity clusters based on the Tiny College example used throughout the content of this book.

When using entity clusters, the key attributes of the combined entities are no longer available. Without the key attributes, primary key inheritance rules change. In turn, the change in the inheritance rules can have undesirable consequences, such as changes in relationships—from identifying to nonidentifying or vice versa—and the loss of foreign key attributes from some entities. To eliminate those problems, the general rule is to avoid the display of attributes when entity clusters are used.

Read

Database Systems: Design, Implementation, and Management, 13th Edition Carlos Coronel and Steven Morris, Chapter Five page 175-176.

Entity Integrity: Selecting Primary Keys

Arguably, the most important characteristic of an entity is its primary key, which uniquely identifies each entity instance. The primary key's function is to guarantee entity integrity. Furthermore, primary keys and foreign keys work together to implement relationships in the relational model. Therefore, the importance of properly selecting the primary key has a direct bearing on the efficiency and effectiveness of database implementation.

Natural Keys and Primary Keys

The concept of a unique identifier is commonly encountered in the real world. A **natural key** or **natural identifier** is a real-world, generally accepted identifier used to distinguish—that is, uniquely identify—real-world objects. As its name implies, a natural key is familiar to end users and forms part of their day-to-day business vocabulary. Usually, if an entity has a natural identifier, a data modeler uses it as the primary key of the entity being modelled. Generally, most natural keys make acceptable primary key identifiers. The next section presents some basic guidelines for selecting primary keys.

Primary Key Guidelines

A **primary key** is the attribute or combination of attributes that uniquely identifies entity instances in an entity set. This section examines that body of documented practices in selecting a primary key. First, you should understand the function of a primary key. Its main function is to *uniquely identify an entity instance or row within a table*. Given a primary key value—that is, the determinant—the relational model can determine the value of all dependent attributes that "describe" the entity. Note that identification and description are separate semantic constructs in the model. The function of the primary key is to guarantee entity integrity, not to "describe" the entity.

Second, primary keys and foreign keys are used to implement relationships among entities. However, the implementation of such relationships is done mostly behind the scenes, hidden from end users. In the real world, end users identify objects based on the characteristics they know about the objects. For example, when shopping at a grocery store, you select products by taking them from a display shelf and reading the labels, not by looking at the stock number. It is wise for database applications to mimic the human selection process as much as possible. Therefore, database applications should let the end user choose among multiple descriptive narratives of different objects, while using primary key values behind the scenes. Keeping those concepts in mind, look at Table 5.3, which summarizes desirable primary key characteristics.

When to Use Composite Primary Keys

In the previous section, you learned about the desirable characteristics of primary keys. For example, you learned that the primary key should use the minimum number of attributes possible. However, that does not mean that composite primary keys are not per- mitted in a model. In fact, composite primary keys are particularly useful in two cases:

- As identifiers of composite entities, in which each primary key combination is allowed only once in the M:N relationship
- As identifiers of weak entities, in which the weak entity has a strong identifying relationship with the parent entity

In both situations, having a strong identifying relationship ensures that the dependent entity can exist only when it is related to the parent entity. In summary, the selection of a composite primary key for composite and weak entity types provides benefits that enhance the integrity and consistency of the model.

When to Use Surrogate Primary Keys

In some instances a primary key doesn't exist in the real world or the existing natural key might not be a suitable primary key. In these cases, it is standard practice to create a surrogate key. A **surrogate key** is a primary key created by the database designer to simplify the identification of entity instances. The surrogate key has no meaning in the user's environment—it exists only to distinguish one entity instance from another (just like any other primary key). One practical advantage of a surrogate key is that because it has no intrinsic meaning, values for it can be generated by the DBMS to ensure that unique values are always provided. For example, consider the case of a park recreation facility that rents rooms for small parties. The manager of the facility keeps track of all events, using a folder with the format shown in Table 5.4.

Given the data shown in Table 5.4, you would model the EVENT entity as follows: EVENT (DATE, TIME_START, TIME_END, ROOM, EVENT_NAME, PARTY_OF) What primary key would you suggest? In this case, there is no simple natural key that could be used as a primary key in the model. Based on the primary key concepts you learned in previous chapters, you might suggest: (DATE, TIME_START, ROOM) or (DATE, TIME_END, ROOM). Taking the first combination may result in a lengthy, four-attribute composite primary key or lengthy primary keys for existence-dependent entities. At this point, you can see that the composite primary key could make the data- base implementation and program coding unnecessarily complex.

TABLE 5.4

DATA USED TO KEEP TRACK OF EVENTS

DATE	TIME_START	TIME_END	ROOM	EVENT_NAME	PARTY_OF
6/17/2018	11:00 a.m.	2:00 p.m.	Allure	Burton Wedding	60
6/17/2018	11:00 a.m.	2:00 p.m.	Bonanza	Adams Office	12
6/17/2018	3:00 p.m.	5:30 p.m.	Allure	Smith Family	15
6/17/2018	3:30 p.m.	5:30 p.m.	Bonanza	Adams Office	12
6/18/2018	1:00 p.m.	3:00 p.m.	Bonanza	Boy Scouts	33
6/18/2018	11:00 a.m.	2:00 p.m.	Allure	March of Dimes	25
6/18/2018	11:00 a.m.	12:30 p.m.	Bonanza	Smith Family	12

The preferred alternative is to use a numeric, single-attribute surrogate primary key. Surrogate primary keys are accepted practice in today's complex data environments. They are especially helpful when there is no natural key, when the selected candidate key has embedded semantic contents, or when the selected candidate key is too long or cumbersome. However, there is a trade-off: if you use a surrogate key, you must ensure that the candidate key of the entity in question performs properly through the "unique index" and "not null" constraints.

Read

Database Systems: Design, Implementation, and Management, 13th Edition Carlos Coronel and Steven Morris, Chapter Five page 176-180.

Design Cases: Learning Flexible Database Design

Data modelling and database design require skills that are acquired through experience. In turn, experience is acquired through practice—regular and frequent repetition, applying the concepts learned to specific and different design problems. This section presents four special design cases that highlight the importance of flexible designs, proper identification of primary keys, and placement of foreign keys.

Design Case 1: Implementing 1:1 Relationships

Foreign keys work with primary keys to properly implement relationships in the relational model. The basic rule is very simple: put the primary key of the "one" side (the parent entity) on the "many" side (the dependent entity) as a foreign key. However, where do you place the foreign key when you are working with a 1:1 relationship? For example, take the case of a 1:1 relationship between EMPLOYEE and DEPARTMENT based on the business rule "one EMPLOYEE is the manager of one DEPARTMENT, and one DEPARTMENT is managed by one EMPLOYEE." In that case, there are two options for selecting and placing the foreign key:

1. Place a foreign key in both entities.
2. Place a foreign key in one of the entities. In that case, the primary key of one of the two entities appears as a foreign key in the other entity.

As a designer, you must recognize that 1:1 relationships exist in the real world; therefore, they should be supported in the data model. In fact, a 1:1 relationship is used to ensure that two entity sets are not placed in the same table. In other words, EMPLOYEE and DEPARTMENT are clearly separate and unique entity types that do not belong together in a single entity. If you grouped them together in one entity, what would you name that entity?

Design Case 2: Maintaining History of Time-Variant Data

Company managers generally realize that good decision making is based on the information generated through the data stored in databases. Such data reflects both current and past events. Company managers use the data stored in databases to answer questions. In other words, the data stored in databases reflects not only current data but also historic data.

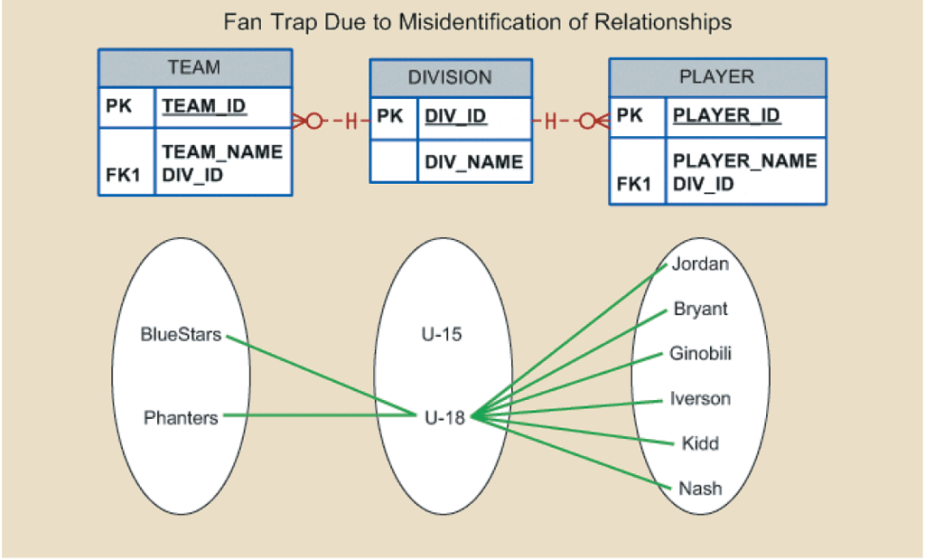
Normally, data changes are managed by replacing the existing attribute value with the new value, without regard to the previous value. However, in some situations the history of values for a given attribute must be preserved. From a data-modelling point of view, **time-variant data** refers to data whose values change over time and for which you must keep a history of the data changes. You could argue that all data in a database is subject to change over time and is therefore time variant. However, some attribute values, such as your date of birth or your Social Security number, are not time variant. On the other hand, attributes such as your student GPA or your bank account balance are subject to change over time. Sometimes the data changes are externally originated and event driven, such as a product price change. On other occasions, changes are based on well-defined schedules, such as the daily stock quote "open" and "close" values. The storage of time-variant data requires changes in the data model; the type of change depends on the nature of the data. Some, time-variant data is equivalent to having a multivalued attribute in your entity. To model this type of time-variant data, you must create a new entity in a 1:M relationship with the original entity. This new entity will contain the new value, the date of the change, and any other attribute that is pertinent to the event being modelled.

Design Case 3: Fan Traps

Creating a data model requires proper identification of the data relationships among entities. However, due to miscommunication or incomplete understanding of the business rules or processes,

it is not uncommon to misidentify relationships among entities. Under those circumstances, the ERD may contain a design trap. A **design trap** occurs when a relationship is improperly or incompletely identified and is therefore represented in a way that is not consistent with the real world. The most common design trap is known as a **fan trap**, which occurs when you have one entity in two 1:M relationships to other entities, thus producing an association among the other entities that is not expressed in the model. For example, assume that the JCB basketball league has many divisions. Each division has many players, and each division has many teams. Given those "incomplete" business rules, you might create an ERD that looks like the one in Figure 5.12.

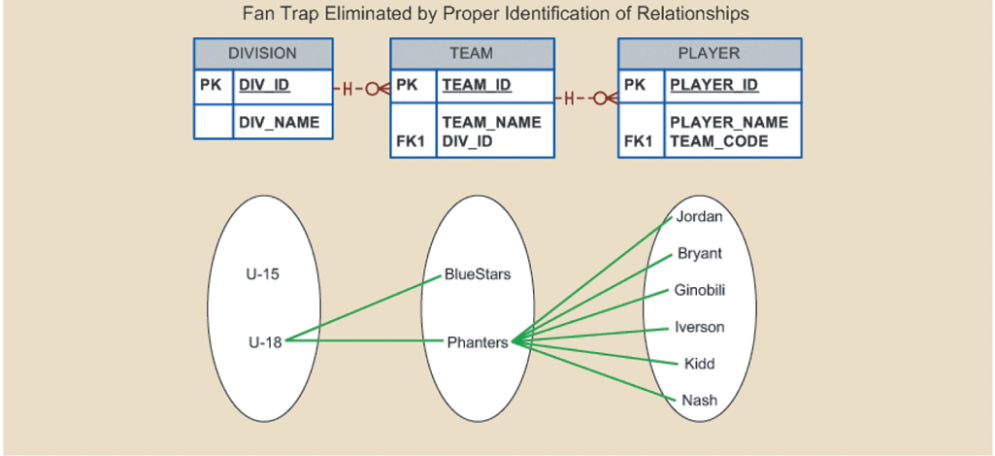
FIGURE 5.12 INCORRECT ERD WITH FAN TRAP PROBLEM



As you can see in Figure 5.12, DIVISION is in a 1:M relationship with TEAM and in a 1:M relationship with PLAYER. Although that representation is semantically correct, the relationships are not properly identified, and we thus have a fan trap.

Figure 5.13 shows the correct ERD after the fan trap has been eliminated.

FIGURE 5.13 CORRECTED ERD AFTER REMOVAL OF THE FAN TRAP



Design Case 4: Redundant Relationships

Although redundancy is often good to have in computer environments (multiple back-ups in multiple places, for example), redundancy is seldom good in the database environment as it may result in data anomalies. Redundant relationships occur when there are multiple relationship paths between related entities. The main concern with redundant relationships is that they remain consistent across the model.

However, it is important to note that some designs use redundant relationships to simplify the design. A specific example of a redundant relationship is represented in Figure 5.14. In Figure 5.14, note the transitive 1:M relationship between DIVISION and PLAYER through the TEAM entity set. Therefore, the relationship that connects DIVISION and PLAYER is redundant, for all practical purposes. In that case, the relationship could be safely deleted without losing any information-generation capabilities in the model.

Read

Database Systems: Design, Implementation, and Management, 13th Edition Carlos Coronel and Steven Morris, Chapter Five page 180-187.

Chapter Summary/Review

In this chapter, you learned:

- 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.
- Keys are central to the use of relational tables. A key can be classified as a super-key, 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 combination of attributes that uniquely identifies all remaining attributes found in any given row.
- Although 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.
- The relational model supports several relational algebra functions, including SELECT, PROJECT, JOIN, INTERSECT, UNION, DIFFERENCE, PRODUCT, and DIVIDE. Understanding the basic mathematical forms of these functions gives a broader understanding of the data manipulation options.
- A relational database performs much of the data manipulation work behind the scenes.
- Once you know the basics of relational databases, you can concentrate on design. Good design begins by identifying appropriate entities and their attributes and then the relationships among the entities.
- The ERM uses ERDs to represent the conceptual database as viewed by the end user. The ERM's main components are entities, relationships, and attributes.
- Connectivity describes the relationship classification (1:1, 1:M, or M:N). Cardinality expresses the specific number of entity occurrences associated with an occurrence of a related entity.
- In the ERM, an M:N relationship is valid at the conceptual level. However, when implementing the ERM in a relational database, the M:N relationship must be mapped to a set of 1:M relationships through a composite entity.
- ERDs may be based on many different ERMs. However, regardless of which model is selected, the modelling logic remains the same.
- Unified Modelling Language (UML) class diagrams are used to represent the static data structures in a data model. The UML class diagrams can be used to depict data models at the conceptual or implementation abstraction levels.
- Database designers are often forced to make design compromises. Those compromises are required when end users have vital transaction-speed and information requirements that prevent the use of "perfect" modelling logic and adherence to all modelling conventions.
- The extended entity relationship (EER) model adds semantics to the ER model via entity super-types, subtypes, and clusters. An entity super-type is a generic entity type that is related to one or more entity subtypes.
- A specialization hierarchy depicts the arrangement and relationships between entity super-types and entity subtypes. There are basically two approaches to developing a specialization hierarchy of entity super-types and subtypes: specialization and generalization.
- An entity cluster is a "virtual" entity type used to represent multiple entities and relationships in the ERD.
- Natural keys are identifiers that exist in the real world. Natural keys sometimes make good primary keys, but not always. Primary keys must have unique values, they should be non-intelligent, they must not change over time, and they are preferably numeric and composed of a single attribute.

- Composite keys are useful to represent M:N relationships and weak (strong identifying) entities.
- Surrogate primary keys are useful when there is no natural key that makes a suitable primary key, when the primary key is a composite primary key with multiple data types, or when the primary key is too long to be usable.
- In a 1:1 relationship, place the PK of the mandatory entity as a foreign key in the optional entity, as an FK in the entity that causes the fewest nulls, or as an FK where the role is played.
- Time-variant data refers to data whose values change over time and require that you keep a history of data changes. To maintain the history of time-variant data, you must create an entity that contains the new value, the date of change, and any other time-relevant data.
- A fan trap occurs when you have one entity in two 1:M relationships to other entities, and there is an association among the other entities that is not expressed in the model. Redundant relationships occur when there are multiple relationship paths between related entities.

Review Questions

1. What is the difference between a database and a table?
2. Why are entity integrity and referential integrity important in a database?
3. Which relational algebra operators can be applied to a pair of tables that are not union-compatible?
4. What is a composite entity, and when is it used?
5. What is a recursive relationship? Give an example.
6. What three (often conflicting) database requirements must be addressed in data-base design?
7. What is a specialization hierarchy?
8. What is a disjoint subtype? Give an example.
9. Under what circumstances are composite primary keys appropriate?

Problems

Use Figure Q3.8 to answer Questions 10–12.

FIGURE Q3.8 THE CH03_COLLEGEQUE DATABASE TABLES

Database name: Ch03_CollegeQue

Table name: STUDENT

STU_CODE	PROF_CODE
100278	
128569	2
512272	4
531235	2
531268	
553427	1

Table name: PROFESSOR

PROF_CODE	DEPT_CODE
1	2
2	6
3	6
4	4

10. Using the STUDENT and PROFESSOR tables, illustrate the difference between a natural join, an equijoin, and an outer join.
11. Create the basic ERD for the database shown in Figure Q3.8.
12. Create the relational diagram for the database shown in Figure Q3.8.

Use the database shown in Figure Q3.22 to answer Questions 13–17.

13. Identify the primary keys.
14. Identify the foreign keys.

15. Create the ERM.

16. Create the relational diagram to show the relationship between DIRECTOR and PLAY.

17. Suppose you wanted quick lookup capability to get a listing of all plays directed by a given director. Which table would be the basis for the INDEX table, and what would be the index key?

18. Tiny College wants to keep track of the history of all its administrative appoint-ments, including dates of appointment and dates of termination. (Hint: Time-variant data is at work.) The Tiny College chancellor may want to know how many deans worked in the College of Business between January 1, 1960, and January 1, 2018, or who the dean of the College of Education was in 1990. Given that infor-mation, create the complete ERD that contains all primary keys, foreign keys, and main attributes.

Review Questions (MCQ)

1. A fan trap occurs when

    a. You have one entity in two 1:1 relationships to other entities, and there is an association among the other entities that is not expressed in the model.
    b. You have one entity in two 1:M relationships to other entities, and there is no association among the other entities that is not expressed in the model.
    c. You have one entity in two N:M relationships to other entities, and there is no association among the other entities that is not expressed in the model.
    d. You have one entity in two 1:M relationships to other entities, and there is an association among the other entities that is not expressed in the model.

2. _____ refers to data whose values change over time and require that you keep a history of data changes.

    a. Date variant data
    b. Time specific data
    c. Time shifting data
    d. None of the above

3. Surrogate primary keys are useful in which of the following cases?

    a. When there is no natural key that makes a suitable primary key
    b. When the primary key is a composite primary key with multiple data types
    c. When the primary key is too long to be usable.
    d. All of the above

Case Study/Project

Case Study 2.1

During peak periods, Temporary Employment Corporation (TEC) places tempo-rary workers in companies. TEC's manager gives you the following description of the business:

- TEC has a file of candidates who are willing to work.
- Any candidate who has worked before has a specific job history. (Naturally, no job history exists if the candidate has never worked.) Each time the candidate works, one additional job history record is created.
- Each candidate has earned several qualifications. Each qualification may be earned by more than one candidate. (For example, more than one candidate may have earned a Bachelor of Business Administration degree or a Microsoft Net-work Certification, and clearly a candidate may have earned both a BBA and a Microsoft Network Certification.)
- TEC offers courses to help candidates improve their qualifications.
- Every course develops one specific qualification; however, TEC does not offer a course for every qualification. Some qualifications are developed through multiple courses.
- Some courses cover advanced topics that require specific qualifications as pre-requisites. Some courses cover basic topics that do not require any prerequisite qualifications. A course can have several prerequisites. A qualification can be a prerequisite for more than one course.
- Courses are taught during training sessions. A training session is the presentation of a single course. Over time, TEC will offer many training sessions for each course; however, new courses may not have any training sessions scheduled right away.
- Candidates can pay a fee to attend a training session. A training session can accommodate several candidates, although new training sessions will not have any candidates registered at first.
- TEC also has a list of companies that request temporaries.
- Each time a company requests a temporary employee, TEC makes an entry in the Openings folder. That folder contains an opening number, a company name, required qualifications, a starting date, an anticipated ending date, and hourly pay.
- Each opening requires only one specific or main qualification.
- When a candidate matches the qualification, the job is assigned, and an entry is made in the Placement Record folder. The folder contains such information as an opening number, candidate number, and total hours worked. In addition, an entry is made in the job history for the candidate.
- An opening can be filled by many candidates, and a candidate can fill many openings.
- TEC uses special codes to describe a candidate's qualifications for an opening. The list of codes is shown in Table P4.12.

TABLE P4.12

CODES FOR PROBLEM 12

CODE	DESCRIPTION
SEC-45	Secretarial work; candidate must type at least 45 words per minute
SEC-60	Secretarial work; candidate must type at least 60 words per minute
CLERK	General clerking work
PRG-VB	Programmer, Visual Basic
PRG-C++	Programmer, C++
DBA-ORA	Database Administrator, Oracle
DBA-DB2	Database Administrator, IBM DB2
DBA-SQLSERV	Database Administrator, MS SQL Server
SYS-1	Systems Analyst, level 1
SYS-2	Systems Analyst, level 2
NW-NOV	Network Administrator, Novell experience
WD-CF	Web Developer, ColdFusion

TEC's management wants to keep track of the following entities: COMPANY, OPENING, QUALIFICATION, CANDIDATE, JOB_HISTORY, PLACEMENT, COURSE, and SESSION.

Given that information, do the following:

a) Draw the Crow's Foot ERDs for this enterprise.

b) Identify all necessary relationships.

c) Identify the connectivity for each relationship.

d) Identify the mandatory and optional dependencies for the relationships.

e) Resolve all M:N relationships.

LEARNING OUTCOMES

After reading this Section of the guide, the learner should be able to:

- Explain normalization and its role in the database design process
- Identify and describe each of the normal forms: 1NF, 2NF, 3NF, BCNF and 4NF
- Explain how normal forms can be transformed from lower normal forms to higher normal forms
- Apply normalization rules to evaluate and correct table structures
- Identify situations that require de-normalization to generate information efficiently
- Use a data-modelling checklist to check that the ERDs meets a set of minimum requirements

3.1 Database Tables and Normalization

Having good relational database software is not enough to avoid the data redundancy discussed in Chapter 1, Database Systems. If the database tables are designed as though they are files in a file system, the relational database management system (RDBMS) never has a chance to demonstrate its superior data-handling capabilities. The table is the basic building block of database design. Consequently, the table's structure is of great interest.

How do you recognize a poor table structure, and how do you produce a good table? The answer to both questions involves normalization. **Normalization** is a process for evaluating and correcting table structures to minimize data redundancies, thereby reducing the likelihood of data anomalies.

The normalization process involves assigning attributes to tables based on the concepts of determination and functional dependency. It works through a series of stages called normal forms. The first three stages are described as first normal form (1NF), second normal form (2NF), and third normal form (3NF). From a structural point of view, 2NF is better than 1NF, and 3NF is better than 2NF.

For most purposes in business database design, 3NF is as high as you need to go in the normalization process. However, even higher-level normal forms exist, including: Boyce-Codd normal form (BCNF), fourth normal form (4NF), fifth normal form (5NF) and domain-key normal form (DKNF). However, normal forms such as the 5NF and DKNF are not likely to be encountered in a business environment and are mainly of theoretical interest.

Furthermore, a successful design must also consider end-user demand for fast performance. Hence, at times you may need to de-normalize some portions of a database design to meet performance requirements. **Denormalization** produces a lower normal form; that is, a 3NF will be converted to a 2NF through denormalization. However, the price you pay for increased performance through denormalization is greater data redundancy.

Read

Database Systems: Design, Implementation, and Management, 13th Edition Carlos Coronel and Steven Morris, Chapter Six page 200-200.

The Need for Normalization

Normalization is typically used by database designers in conjunction with the ERM; when designing a new database structure based on the business requirements of the end users and to analyse the relationships among the attributes within each entity, of an existing database, and determine if the structure can be improved through normalization. In either case, the normalization process is the same.

Commonly, you start by defining the business rules and data constraints, identifying the functional dependencies, entities, and attributes using the techniques you learned in previous chapters. Then, you apply normalization concepts to validate and further refine the model.

The main goal of normalization is to eliminate data anomalies by eliminating unnecessary or unwanted data redundancies. This goal is achieved by using the concept of functional dependencies to identify which attribute (or set of attributes) determines other attributes.

To get a better idea of the normalization process, consider the simplified reporting activities of a construction company that manages several building projects. The base data in Figure 6.1 is organized around projects, with each project having a single row to represent the data associated with that project. The base data shows that a project has multiple employees assigned to it.

FIGURE 6.1 BASE DATA FOR A CONSTRUCTION COMPANY REPORT

Table name: RPT_FORMAT Database name: Ch06_ConstructCo

PROJ_NUM	PROJECT_NAME	EMP_NUMBER	EMP_NAME	JOB_CLASS	CHARGE_HOUR	HOURS_BILLED
15	Evergreen	103,101,105, 106, 102	June E. Arbough, John G. News, Alice K. Johnson *, William Smithfield, David H. Senior	Elec. Engineer, Database Designer, Database Designer, Programmer, System Analyst	85.5, 105., 105., 35.75, 98.75	23.8, 19.4, 35.7, 12.6, 23.8
18	Amber Wave	114, 118, 104, 112	Annelise Jones, James J. Frommer, Anne K. Ramoras *, Darlene M. Smithson	Applications Designer, General Support, Systems Analyst, DSS Analyst	48.1, 18.36, 96.75, 45.95	25.6, 45.3, 32.4, 45.
22	Rolling Tide	105, 104, 113, 111, 106	Alice K. Johnson, Anne K. Ramoras, Delbert K. Joenbrood *, Geoff B. Wabash, William Smithfield	DB Designer, Systems Analyst, Applications Designer, Clerical Support, Programmer	105., 96.75, 48.1, 26.87, 35.75	65.7, 48.4, 23.6, 22., 12.8
25	Star Light	107, 115, 101, 114, 108, 118, 112	Maria D. Alonzo, Travis B. Bawangi, John G. News *, Annelise Jones, Ralph B. Washington, James J. Frommer, Darlene M. Smithson	Programmer, Systems Analyst, Database Design, Applications Designer, Systems Analyst, General Support, DSS Analyst	35.75, 96.75, 105., 48.1, 96.75, 18.36, 45.95	25.6, 45.8, 56.3, 33.1, 23.6, 30.5, 41.4

Note that the data in Figure 6.1 is **unnormalized data** and it does not conform to relational table requirements and therefore is not suitable to handle data updates well. Consider the following deficiencies:

- The data structure invites data inconsistencies.
- The data structure contains several multivalued attributes that make data management tasks very difficult.
- Employee data is redundant in the table because employees can work on multiple projects. Adding, updating, and deleting data are likely to be very cumbersome using this structure.

Clearly, this data structure yields data inconsistencies. A report generated might yield varying results depending on which data anomaly has occurred. Such reporting anomalies cause a multitude of problems for managers—and cannot be fixed through application programming. These data integrity, data redundancy, and data inconsistency problems must be addressed during database design. The next section walks you through the normalization process used to minimize redundancies and eliminate data anomalies.

Read

Database Systems: Design, Implementation, and Management, 13th Edition Carlos Coronel and Steven Morris, Chapter Six page 200-203

3.2 The Normalization Process

In this section, you learn how to use normalization to produce a set of normalized relations (tables) that will be used to generate the required information. The objective of normalization is to ensure tables that have the following characteristics:

- Each relation (table) represents a single subject.
- Each row/column intersection contains only one value and not a group of values.
- No data item will be unnecessarily stored in more than one table.
- All nonprime attributes in a relation (table) are dependent on the primary key.
- Each relation (table) has no insertion, update, or deletion anomalies, which ensures the integrity and consistency of the data.

To accomplish these objectives, the normalization process takes you through steps that lead to successively higher normal forms. The most common normal forms and their basic characteristics are listed in Table 6.2.

TABLE 6.2

NORMAL FORMS

NORMAL FORM	CHARACTERISTIC	SECTION
First normal form (1NF)	Table format, no repeating groups, and PK identified	6-3a
Second normal form (2NF)	1NF and no partial dependencies	6-3b
Third normal form (3NF)	2NF and no transitive dependencies	6-3c
Boyce-Codd normal form (BCNF)	Every determinant is a candidate key (special case of 3NF)	6-6a
Fourth normal form (4NF)	3NF and no independent multivalued dependencies	6-6b

Although normalization is typically presented from the perspective of candidate keys, this initial discussion assumes that each table has only one candidate key; therefore, that candidate key is the primary key.

Before outlining the normalization process, it is a good idea to review the concepts of determination and functional dependence. Table 6.3 summarizes the main concepts.

TABLE 6.3

FUNCTIONAL DEPENDENCE CONCEPTS

CONCEPT	DEFINITION
Functional dependence	The attribute *B* is fully functionally dependent on the attribute *A* if each value of *A* determines one and only one value of *B*. Example: PROJ_NUM → PROJ_NAME (read as *PROJ_NUM functionally determines PROJ_NAME*) In this case, the attribute PROJ_NUM is known as the determinant attribute, and the attribute PROJ_NAME is known as the dependent attribute.
Functional dependence (generalized definition)	Attribute *A* determines attribute *B* (that is, *B* is functionally dependent on *A*) if all (generalized definition) of the rows in the table that agree in value for attribute *A* also agree in value for attribute *B*.
Fully functional dependence (composite key)	If 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*.

The normalization process works one relation at a time, identifying the dependencies on that relation and normalizing the relation. Normalization starts by identifying the dependencies of a given relation and progressively breaking up the relation (table) into a set of new relations (tables) based on the identified dependencies.

Two types of functional dependencies that are of special interest in normalization are partial dependencies and transitive dependencies. A **partial dependency** exists when there is a functional dependence in which the determinant is only part of the primary key. For example, if (A, B) → (C,

D), B → C, and (A, B) is the primary key, then the functional dependence B → C is a partial dependency because only part of the primary key (B) is needed to determine the value of C.

A **transitive dependency** exists when there are functional dependencies such that X →Y, Y → Z, and X is the primary key. In that case, the dependency X → Z is a transitive dependency because X determines the value of Z via Y. An effective way to identify transitive dependencies is: they occur only when a functional dependence exists among nonprime attributes. To address the problems related to transitive dependencies, changes to the table structure are made based on the functional dependence that signals the transitive dependency's existence. Therefore, to simplify the description of normalization, from this point forward the signalling dependency will be called the transitive dependency.

Conversion to First Normal Form (1NF)

Because the relational model views data as part of a table or a collection of tables in which all key values must be identified, the data depicted in Figure 6.1 might not be stored as shown. Note that Figure 6.1 contains what is known as repeating groups. A **repeating group** derives its name from the fact that a group of multiple entries of the same or multiple types can exist for any single key attribute occurrence.

Normalizing the table structure will reduce the data redundancies. Normalisation starts with a simple three-step procedure:

Step 1: Eliminate the Repeating Groups Start by presenting the data in a tabular format, where each cell has a single value and there are no repeating groups. This change converts the table in Figure 6.1 to 1NF as shown in Figure 6.2.

Step 2: Identify the Primary Key The layout in Figure 6.2 represents more than a mere cosmetic change. Even a casual observer will note that PROJ_NUM is not an adequate primary key because the project number does not uniquely identify each row. To maintain a proper primary key that will uniquely identify any attribute value, the new key must be composed of a combination of PROJ_NUM and EMP_NUM.

Step 3: Identify All Dependencies The identification of the PK in Step 2 means that you have already identified the following dependency:

PROJ_NUM, EMP_NUM S PROJ_NAME, EMP_NAME, JOB_CLASS, CHG_HOUR, HOURS

That is, the PROJ_NAME, EMP_NAME, JOB_CLASS, CHG_HOUR, and HOURS values are all dependent on—they are determined by—the combination of PROJ_NUM and EMP_NUM.

Achieving 1NF is insufficient to address all the anomalies that existed in the original structure. 1NF has dealt with the repeating groups and ensured that our table conforms to the requirements for a relational table. However, anomalies remain. The anomalies that remain exist because there are additional dependencies.

By further studying the data in Figure 6.2, you can see that knowing the job classification means knowing the charge per hour for that job classification. Therefore, you can identify one last dependency:

JOB_CLASS → CHG_HOUR

However, this dependency exists between two nonprime attributes; therefore, it is a signal that a transitive dependency exists, and we will refer to it as a transitive dependency. The dependencies you have just examined can also be depicted with the help of the diagram shown in Figure 6.3.

Because such a diagram depicts all dependencies found within a given table structure, it is known as a **dependency diagram**.

As you examine Figure 6.3, note the following features of a dependency diagram:

1. The primary key attributes are bold, underlined, and in a different colour.
2. The arrows above the attributes indicate all desirable dependencies—that is, dependencies based on the primary key. In this case, note that the entity's attributes are dependent on the combination of PROJ_NUM and EMP_NUM.
3. The arrows below the dependency diagram indicate less desirable dependencies. Two types of such dependencies exist:
    a. *Partial dependencies*. A dependency based on only a part of a composite primary key is a **partial dependency**.
    b. *Transitive dependencies*. A **transitive dependency** exists when a functional dependency exists only among nonprime attributes. Transitive dependencies yield data anomalies.

Figure 6.3 includes the relational schema for the table in 1NF and a textual notation for each identified dependency. All relational tables satisfy the 1NF requirements, but it still has undesirable problems. A table that contains partial dependencies is still subject to data redundancies, and therefore to various anomalies. Our example still has the following anomalies:

- Update anomalies.
- Insertion anomalies.
- Deletion anomalies.

The data redundancies occur because every row entry requires duplication of data. Such duplication of effort is very inefficient, and it helps create data anomalies. Such data anomalies violate the relational database's integrity and consistency rules.

Conversion to Second Normal Form (2NF)

Conversion to 2NF occurs only when the 1NF has a composite primary key. If the 1NF has a single-attribute primary key, then the table is automatically in 2NF. The 1NF-to-2NF conversion is simple. Starting with the 1NF format displayed in Figure 6.3, you take the following steps:

Step 1: Make New Tables to Eliminate Partial Dependencies For each component of the primary key that acts as a determinant in a partial dependency, create a new table with a copy of that component as the primary key. While these components are placed in the new tables, it is important that they also remain in the original table as well. The determinants must remain in the original table because they will be the foreign keys for the relationships needed to relate these new tables to the original table. To construct the revised dependency diagram, write each key component on a separate line and then write the original (composite) key on the last line.

Step 2: Reassign Corresponding Dependent Attributes Use Figure 6.3 to determine attributes that are dependent in the partial dependencies. The dependencies for the original key components are found by examining the arrows below the dependency diagram shown in Figure 6.3. The attributes that are dependent in a partial dependency are removed from the original table and placed in the new table with the dependency's determinant. In other words, the three tables that result from the conversion to 2NF are given appropriate names.

The results of Steps 1 and 2 are displayed in Figure 6.4. At this point, most of the anomalies discussed earlier have been eliminated.

Figure 6.4 still shows a transitive dependency, which can generate anomalies.

Conversion to Third Normal Form (3NF)

The data anomalies created by the database organization shown in Figure 6.4 are easily eliminated by completing the following two steps:

Step 1: Make New Tables to Eliminate Transitive Dependencies For every transitive dependency, write a copy of its determinant as a primary key for a new table. A **determinant** is any attribute whose value determines other values within a row. If you have three different transitive dependencies, you will have three different determinants. As with the conversion to 2NF, it is important that the determinant remain in the original table to serve as a foreign key.

Step 2: Reassign Corresponding Dependent Attributes Using Figure 6.4, identify the attributes that are dependent on each determinant identified in Step 1. Place the dependent attributes in the new tables with their determinants and remove them from their original tables.

EMP_NUM → EMP_NAME, JOB_CLASS

Draw a new dependency diagram to show all the tables you have defined in Steps 1 and 2. Name the table to reflect its contents and function. Check all the tables to make sure that each table has a determinant and that no table contains inappropriate dependencies. When you have completed these steps, you will see the results in Figure 6.5 and the tables are said to be in third normal form (3NF).

It is interesting to note the similarities between resolving 2NF and 3NF problems. No matter whether the "problem" dependency is a partial dependency or a transitive dependency, the solution is the same: create a new table for each problem dependency. The determinant of the problem dependency remains in the original table and is placed as the primary key of the new table. The dependents of the problem dependency are removed from the original table and placed as nonprime attributes in the new table.

In the above examples each table has only one candidate key, which is the primary key. If a table has multiple candidate keys, then the overall process remains the same, but there are additional considerations.

The existence of multiple candidate keys can also influence the identification of transitive dependencies. Previously, a transitive dependency was defined to exist when one nonprime attribute determined another nonprime attribute. In the presence of multiple candidate keys, the definition of a nonprime attribute as an attribute that is not a part of any candidate key is critical. If the determinant of a functional dependence is not the primary key but is a part of another candidate key, then it is not a nonprime attribute and does not signal the presence of a transitive dependency.

Read

Database Systems: Design, Implementation, and Management, 13th Edition Carlos Coronel and Steven Morris, Chapter Six page 203-203.

3.3 Improving the Design

Now that the table structures have been cleaned up to eliminate the troublesome partial and transitive dependencies, you can focus on improving the database's ability to provide information and on enhancing its operational characteristics. Following, you will learn about the different types of issues you need to address to produce a well normalized set of tables. Remember that normalization cannot, by itself, produce good designs. Thus, other considerations need to be made to improve the overall design of the database. These include:

Evaluating PK Assignments. Unfortunately, it is too easy to make data-entry errors that lead to referential integrity violations if a key attribute needs to be entered with every new record. For example, each time a new employee is entered in the EMPLOYEE table, a JOB_CLASS value must be entered. In such cases, it would be better to add a new attribute to create a unique identifier, and thus a new primary key would be chosen.

Evaluating Naming Conventions. It is best to adhere to the naming conventions outlined in the discussion about Data Models (Section 1.2 of learner guide). Therefore, to make sure that all tables conform, some table names may be changed after the completion of the normalization process.

Refining Attribute Atomicity. It is generally good practice to pay attention to the atomicity requirement. An atomic attribute is one that cannot be further subdivided. Such an attribute is said to display atomicity. By improving the degree of atomicity, you also gain querying flexibility.

Identifying New Attributes. Take the EMPLOYEE table for an example, if it were used in a real-world environment, several other attributes would have to be added. These attributes would be useful in data processing to generate information that be used in decision making. The same principle must be applied to all other tables in your design.

Identifying New Relationships. In carefully studying the business rules and past reports, new relationships may be identified. In identifying these new relationships, the designer must then take care to place the right attributes in the right tables by using normalization principles.

Refining Primary Keys as Required for Data Granularity. **Granularity** refers to the level of detail represented by the values stored in a table's row. Data stored at its lowest level of granularity is said to be atomic data, as explained earlier.

Maintaining Historical Accuracy. The addition of certain attributes to a table may be crucial to maintaining the historical accuracy of the table's data. For example, writing the charge per hour on the ASSIGNMENT table is crucial for maintaining its history.

Evaluating Using Derived Attributes. Finally, you can use a derived attribute in a table to store a value computed using values from other attributes. This creates a transitive dependency. From a system functionality point of view, derived attribute values can be calculated when they are needed to write reports or invoices. However, storing the derived attribute in the table makes it easy to write the application software to produce the desired results. Also, if many transactions must be reported and/or summarized, the availability of the derived attribute will save reporting time.

Read

Database Systems: Design, Implementation, and Management, 13th Edition Carlos Coronel and Steven Morris, Chapter Six, page 213-217.

Surrogate Key Consideration

Although this design meets the vital entity and referential integrity requirements, the designer must still address some concerns. For example, a composite primary key might become too cumbersome to use as the number of attributes grows. Or, a primary key attribute might simply have too much descriptive content to be usable. When the primary key is considered unsuitable for some reason, designers use surrogate keys.

At the implementation level, a surrogate key is a system-defined attribute generally created and managed via the DBMS. Usually, a system-defined surrogate key is numeric, and its value is automatically incremented for each new row. However, in some cases, the best solution might be to add a new externally defined attribute—such as a stub, voucher, or ticket number—to ensure uniqueness. In any case, frequent data audits would be appropriate.

Read

Database Systems: Design, Implementation, and Management, 13th Edition Carlos Coronel and Steven Morris, Chapter Six page 217-218

Higher-Level Normal Forms

Tables in 3NF will perform suitably in business transactional databases. However, higher normal forms are sometimes useful. In this section, you will learn about a special case of 3NF, known as Boyce-Codd normal form, and about fourth normal form (4NF).

The Boyce-Codd Normal Form

A table is in Boyce-Codd normal form (BCNF) when every determinant in the table is a candidate key. Clearly, when a table contains only one candidate key, the 3NF and the BCNF are equivalent. In other words, BCNF can be violated only when the table contains more than one candidate key. Remember, however, that multiple candidate keys are always possible, and normalization rules focus on candidate keys, not just the primary key. Consider the table structure shown in Figure 6.7.

The CLASS table has two candidate keys:

• CLASS_CODE

• CRS_CODE + CLASS_SECTION

The table is in 2NF because it is in 1NF and there are no partial dependencies on either candidate key. However, the composite candidate key of CRS_CODE + CLASS_SECTION could potentially have a partial dependency, so 2NF must be evaluated for that candidate key. In this case, there are no partial dependencies involving the composite key. Finally, the table is in 3NF because there are no transitive dependencies.

So, how can a table be in 3NF and not be in BCNF? To answer that question, you must keep in mind that a transitive dependency exists when one non-prime attribute is dependent on another nonprime attribute.

Hence in a case in which one key attribute is the determinant of another key attribute, the table fails to meet the BCNF requirements (see Figure 6.8) because BCNF requires that every determinant in the table be a candidate key.

To convert the table structure in Figure 6.8 into table structures that are in 3NF and in BCNF, first change the primary key to A + C. This change is appropriate because the dependency C → B means that C is effectively a superset of B. At this point, the table is in 1NF because it contains a partial dependency, C → B. Next, follow the standard decomposition procedures to produce the results shown in Figure 6.9. To see how this procedure can be applied to an actual problem, examine the sample data in Table 6.5. Table 6.5 reflects the following conditions:

1. Each CLASS_CODE identifies a class uniquely. This condition illustrates the case in which a course might generate many classes.
2. A student can take many classes.
3. A staff member can teach many classes, but each class is taught by only one staff member.

Fourth Normal Form (4NF)

You might encounter poorly designed databases, or you might be asked to convert spreadsheets into a database format in which multiple multivalued attributes exist.

There is a problem with the tables in Figure 6.11. The attributes ORG_CODE and ASSIGN_NUM each may have many different values. In normalization terminology, this situation is referred to as a multivalued dependency, which occurs when one key determines multiple values of two other attributes and those attributes are independent of each other.

The presence of a multivalued dependency means that if table versions 1 and 2 are implemented, the tables are likely to contain quite a few null values; in fact, the tables do not even have a viable candidate key. Such a condition is not desirable, especially when there are thousands of employees, many of whom may have multiple job assignments and many service activities. Version 3 at least has a PK, but it is composed of all the attributes in the table. In fact, version 3 meets 3NF requirements, yet it contains many redundancies that are clearly undesirable. The solution is to eliminate the problems caused by the multivalued dependency. You do this by creating new tables for the components of the multivalued dependency. In this example, the multivalued dependency is resolved and eliminated by creating ASSIGNMENT and SERVICE_V1 tables. Those tables are said to be in 4NF.

If you follow the proper design procedures illustrated in this book, you should not encounter the problem shown in Figure 6.11. Specifically, the discussion of 4NF is largely academic if you make sure that your tables conform to the following two rules:

1. All attributes must be dependent on the primary key, but they must be independent of each other.
2. No row may contain two or more multivalued facts about an entity.

Read

Database Systems: Design, Implementation, and Management, 13th Edition Carlos Coronel and Steven Morris, Chapter Six page 218-224.

3.4 De-normalization

It is important to remember that the optimal relational database implementation requires that all tables be at least in 3NF. Furthermore, good database design also considers processing (or reporting) requirements and processing speed. The problem with normalization is that as tables are decomposed to conform to normalization requirements, the number of database tables expands. Joining a large number of tables takes additional input/output (I/O) operations and processing logic, thereby reducing system speed. Most relational database systems can handle joins very efficiently. However, rare and occasional circumstances may allow require degree of denormalization, so processing speed can be increased.

Keep in mind that the advantage of higher processing speed must be carefully weighed against the disadvantage of data anomalies. On the other hand, some anomalies are of only theoretical interest. As explained earlier, the problem with denormalized relations and redundant data is that data integrity could be compromised due to the possibility of insert, update, and deletion anomalies. The advice is simple: use common sense during the normalization process.

Furthermore, the database design process could, in some cases, introduce some small degree of redundant data in the model. This, in effect, creates “denormalized” relations. Table 6.6 shows some common examples of data redundancy that are generally found in database implementations.

Also, in some cases a solution in 3NF is practical only if performance is not an issue and there are no other viable processing options. As shown, normalization purity is often difficult to sustain in the modern database environment.

Lower normalization forms occur (and are even required) in specialized databases known as data warehouses. The data warehouse routinely uses 2NF structures in its complex, multilevel, multisource data environment. Although normalization is very important, especially in the so-called production database environment, 2NF is no longer disregarded as it once was.

Although 2NF tables cannot always be avoided, the problem of working with tables that contain partial and/or transitive dependencies in a production database environment should not be minimized. Aside from the possibility of troublesome data anomalies being created, unnormalized tables in a production database tend to suffer from these defects:

- Data updates are less efficient because programs that read and update tables must deal with larger tables.
- Indexing is more cumbersome.
- Unnormalized tables yield no simple strategies for creating virtual tables known as views.

Remember that good design cannot be created in the application programs that use a data- base. Also keep in mind that unnormalized database tables often lead to various data redundancy disasters in production databases, such as the problems examined thus far. In other words, use denormalization cautiously and make sure that you can explain why the unnormalized tables are a better choice in certain situations than their normalized counterparts.

Database Systems: Design, Implementation, and Management, 13th Edition Carlos Coronel and Steven Morris, Chapter Six page 224-227.

Chapter Summary/Review

In this chapter, you have learned:

- Normalization is a technique used to design tables in which data redundancies are minimized. The first three normal forms (1NF, 2NF, and 3NF) are the most common. From a structural point of view, higher normal forms are better than lower normal forms.

- A table is in 1NF when all key attributes are defined, and all remaining attributes are dependent on the primary key. However, a table in 1NF can still contain both partial and transitive dependencies.

- A table is in 2NF when it is in 1NF and contains no partial dependencies. Therefore, a 1NF table is automatically in 2NF when its primary key is based on only a single attribute.

- A table is in 3NF when it is in 2NF and contains no transitive dependencies. Given that definition, the Boyce-Codd normal form (BCNF) is merely a special 3NF case in which all determinant keys are candidate keys.

- A table that is not in 3NF may be split into new tables until all the tables meet the 3NF requirements.

- Normalization is an important part—but only a part—of the design process. As entities and attributes are defined during the ER modelling process, subject each entity (set) to normalization checks and form new entities (sets) as required.

- A table in 3NF might contain multivalued dependencies that produce either numerous null values or redundant data. Therefore, it might be necessary to convert a 3NF table to the fourth normal form (4NF) by splitting the table to remove the multivalued dependencies. Thus, a table is in 4NF when it is in 3NF and contains no multivalued dependencies.

- Tables are sometimes de-normalized to yield less I/O in order to increase processing speed. In the design of production databases, use de-normalization sparingly and cautiously.

- The data-modelling checklist provides a way for the designer to check that the ERD meets a set of minimum requirements.

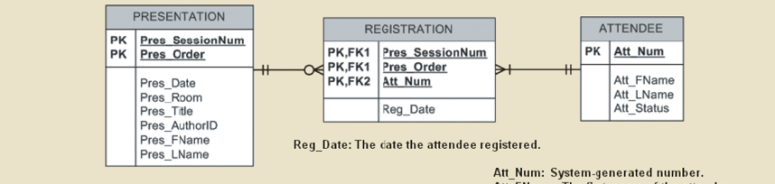
Review Questions

1. What is normalization?
2. When is a table in 2NF?
3. When is a table in BCNF?
4. What is a partial dependency? With what normal form is it associated?
5. Define and discuss the concept of transitive dependency.
6. What is a surrogate key, and when should you use one?

Problems

7. What three data anomalies are likely to be the result of data redundancy? How can such anomalies be eliminated?
8. Using the descriptions of the attributes given in the figure, convert the ERD shown in Figure P6.2 into a dependency diagram that is in at least 3NF.

FIGURE P6.2 PRESENTATION ERD FOR PROBLEM 2



Reg_Date: The date the attendee registered.

Pres_SessionNum: System-generated number.
Pres_Order: Number indicating the order of the presentations during the session.
Pres_Date: The date that the presentation is scheduled to be given.
Pres_Room: The room in which the presentation will be given.
Pres_Title: The title of the presentation.
Pres_AuthorID: System-generated number assigned to presentation authors.
Pres_FName: The first name of the presentation author.
Pres_LName: The last name of the presentation author.

Att_Num: System-generated number.
Att_FName: The first name of the attendee.
Att_LName: The last name of the attendee.
Att_Status: Whether or not the attendee has paid the registration fee.

9. Using the INVOICE table structure shown in Table P6.3, do the following:

TABLE P6.3

ATTRIBUTE NAME	SAMPLE VALUE	SAMPLE VALUE	SAMPLE VALUE	SAMPLE VALUE	SAMPLE VALUE
INV_NUM	211347	211347	211347	211348	211349
PROD_NUM	AA-E3422QW	QD-300932X	RU-995748G	AA-E3422QW	GH-778345P
SALE_DATE	15-Jan-2018	15-Jan-2018	15-Jan-2018	15-Jan-2018	16-Jan-2018
PROD_LABEL	Rotary sander	0.25-in. drill bit	Band saw	Rotary sander	Power drill
VEND_CODE	211	211	309	211	157
VEND_NAME	NeverFail, Inc.	NeverFail, Inc.	BeGood, Inc.	NeverFail, Inc.	ToughGo, Inc.
QUANT_SOLD	1	8	1	2	1
PROD_PRICE	$49.95	$3.45	$39.99	$49.95	$87.75

   a. Write the relational schema, draw its dependency diagram, and identify all dependencies, including all partial and transitive dependencies. You can assume that the table does not contain repeating groups and that an invoice number references more than one product. (Hint: This table uses a composite primary key.)

b. Remove all partial dependencies, write the relational schema, and draw the new dependency diagrams. Identify the normal forms for each table structure you created.
c. Remove all transitive dependencies, write the relational schema, and draw the new dependency diagrams. Also identify the normal forms for each table structure you created.
d. Draw the Crow's Foot ERD.

10. To keep track of office furniture, computers, printers, and other office equipment, the FOUNDIT Company uses the table structure shown in Table P6.5.

TABLE P6.5

ATTRIBUTE NAME	SAMPLE VALUE	SAMPLE VALUE	SAMPLE VALUE
ITEM_ID	231134-678	342245-225	254668-449
ITEM_LABEL	HP DeskJet 895Cse	HP Toner	DT Scanner
ROOM_NUMBER	325	325	123
BLDG_CODE	NTC	NTC	CSF
BLDG_NAME	Nottooclear	Nottooclear	Canseefar
BLDG_MANAGER	I. B. Rightonit	I. B. Rightonit	May B. Next

a. Given that information, write the relational schema and draw the dependency diagram. Make sure that you label the transitive and/or partial dependencies.
b. Write the relational schema and create a set of dependency diagrams that meet 3NF requirements. Rename attributes to meet the naming conventions, and create new entities and attributes as necessary.
c. Draw the Crow's Foot ERD.

Review Questions (MCQ)

1. A table is in 3NF normal form when it is in _____ and there are no transitive dependencies.

   a. BCNF
   b. 3NF
   c. 1NF
   d. 2NF

2. To ensure entity integrity, a primary key may be:
   a. Not NULL
   b. NULL
   c. Both not NULL & NULL
   d. Any value

3. Normalization is a technique used to design tables in which _____ are minimized.

   a. Data independencies
   b. Data redundancies
   c. Data anomalies
   d. All of the above

LEARNING OUTCOMES

After reading this Section of the guide, the learner should be able to:

- Retrieve specified columns of data from a database
- Join multiple tables in a single SQL query
- Restrict data retrievals to rows that match complex criteria
- Aggregate data across groups of rows
- Create subqueries to preprocess data for inclusion in other queries
- Identify and use a variety of SQL functions for string, numeric, and date manipulation
- Explain the key principles in crafting a SELECT query
- Use SQL to create a table manually
- Use SQL to create a copy of a table using a subquery
- Manipulate the structure of existing tables to add, modify, and remove columns and constraints
- Use SQL to do data manipulation (insert, update, and delete rows of data)
- Use SQL to create database views, including updatable views
- Use Procedural Language SQL (PL/SQL) to create triggers, stored procedures, and PL/SQL functions
- Create embedded SQL

4.1 Intro to Structured Query Language (SQL)

Introduction to SQL

Ideally, a database language allows you to create database and table structures, perform basic data management chores (add, delete, and modify), and perform complex queries designed to transform the raw data into useful information. Moreover, a database language must perform such basic functions with minimal user effort, and its command structure and syntax must be easy to learn. Finally, it must be portable; that is, it must conform to some basic standard, so a person does not have to relearn the basics when moving from one RDBMS to another. SQL meets those ideal database language requirements well. SQL functions fit into several broad categories:

- *It is a data manipulation language (DML)*. SQL includes commands to insert, update, delete, and retrieve data within the database tables.
- It is a data definition language (DDL). SQL includes commands to create database objects such as tables, indexes, and views, as well as commands to define access rights to those database objects.
- *It is a transaction control language (TCL)*. The DML commands in SQL are executed within the context of a transaction, which is a logical unit of work composed of one or more SQL statements. SQL provides commands to control the processing of these statements an indivisible unit of work.
- *It is a data control language (DCL)*. Data control commands are used to control access to data objects, such as giving a one user permission to only view a table and giving another user permission to change the data in the table. Common TCL and DCL commands are shown in Table 7.3.

SQL is a nonprocedural language: you merely command what is to be done; you do not have to worry about how. For example, a single command creates the complex table structures required to store and manipulate data successfully; end users and programmers do not need to know the physical data storage format or the complex activities that take place when a SQL command is executed.

The American National Standards Institute (ANSI) prescribes a standard SQL. The ANSI SQL standards are also accepted by the International Organization for Standardization (ISO), a consortium composed of national standards bodies of more than 150 countries. Hence, whether you use Oracle, Microsoft SQL Server, MySQL, IBM DB2, Microsoft Access, or any other well-established RDBMS, a software manual should be sufficient to get you up to speed if you know the material presented in this chapter.

Data Types

The ANSI/ISO SQL standard defines many different data types. A data type is a specification about the kinds of data that can be stored in an attribute. Data types influence queries that retrieve data because there are slight differences in the syntax of SQL and how it behaves during a query that are based on the data type of the column being retrieved.

For now, consider that there are three fundamental types of data: character data, numeric data, and date data. Character data is composed of any printable characters such as alphabetic values, digits, punctuation, and special characters. *Character data* is also often referred to as a “string” because it is a collection of characters threaded together to create the value. *Numeric data* is composed of digits, such that the data has a specific numeric value. *Date data* is composed of date and, occasionally, time values. Although character data may contain digits, the DBMS does not recognize the numeric value of those digits.

SQL Queries

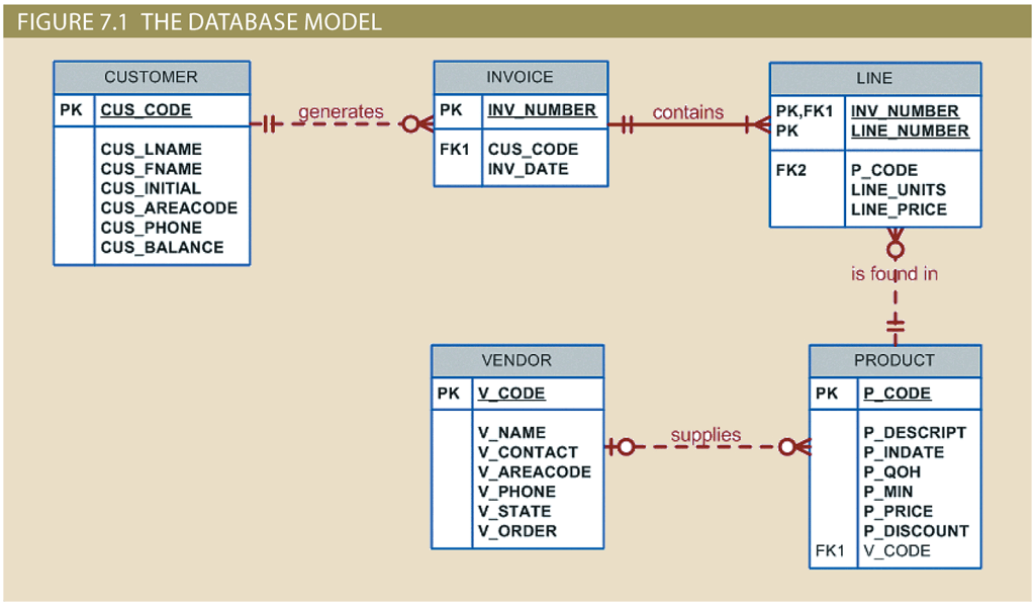
At the heart of SQL is the query. In the SQL environment, the word query covers both questions and actions. Most SQL queries are used to answer questions such as these: "What products currently held in inventory are priced over $100, and what is the quantity on hand for each of those products?" etc. However, many SQL queries are used to perform actions such as adding or deleting table rows or changing attribute values within tables. Still other SQL queries create new tables or indexes. For a DBMS, a *query* is simply a SQL statement that must be executed.

By default, most SQL data manipulation commands operate over an entire table (relation), which is why SQL commands are said to be set-oriented commands. A SQL set-oriented command works over a set of rows. The set may include one or more columns and zero or more rows from one or more tables.

The Database Model

A simple database composed of the following tables is used to illustrate the SQL commands in this chapter: CUSTOMER, INVOICE, LINE, PRODUCT, and VENDOR. This database model is shown in Figure 7.1.

FIGURE 7.1 THE DATABASE MODEL



The database model in Figure 7.1 reflects the following business rules:

- A customer may generate many invoices. Each invoice is generated by one customer.
- An invoice contains one or more invoice lines. Each invoice line is associated with one invoice.
- Each invoice line references one product. A product may be found in many invoice lines. (You can sell more than one hammer to more than one customer.)
- A vendor may supply many products. Some vendors do not yet supply products. For example, a vendor list may include potential vendors.
- If a product is vendor-supplied, it is supplied by only a single vendor.
- Some products are not supplied by a vendor. For example, some products may be produced in-house or bought on the open market.

Except as noted, the database model shown in Figure 7.1 will be used for the queries in the remainder of the chapter. Recall that when an ERD is implemented as a database, each entity becomes a table in the database, and each attribute within an entity becomes a column in that table.

Read

Database Systems: Design, Implementation, and Management, 13th Edition Carlos Coronel and Steven Morris, Chapter Seven, page 245-249.

Basic SELECT Queries

Data retrieval is done in SQL using a SELECT query. When you run a SELECT command on a table, the RDBMS returns a set of one or more rows that have the same characteristics as a relational table. Each clause in a SELECT query performs a specific function. Understanding the function of each clause is key to developing the skills to construct queries to satisfy the reporting needs of the users. The following clauses will be covered in this chapter (although not in this order).

• SELECT—specifies the attributes to be returned by the query

• FROM—specifies the table(s) from which the data will be retrieved

• WHERE—filters the rows of data based on provided criteria

• GROUP BY—groups the rows of data into collections based on sharing the same values in one or more attributes

• HAVING—filters the groups formed in the GROUP BY clause based on provided criteria

• ORDER BY—sorts the final query result rows in ascending or descending order based on the values of one or more attributes.

For a SELECT query to retrieve data from the database, it will require at least a SELECT column list and a FROM clause. The **SELECT** column list specifies the relational projection. The column list allows the programmer to specify which columns should be retrieved by the query and the order in which they should be returned. Only columns specified in the column list will appear in the query result. The **FROM** clause is used to specify the table from which the data will be retrieved. It is common for queries to retrieve data from multiple tables that have been joined together. However, first, we will focus on things that can be done with the column list before we move on to the FROM clause options.

Read

Database Systems: Design, Implementation, and Management, 13th Edition Carlos Coronel and Steven Morris, Chapter Seven, page 249-250.

SELECT Statement Options

The SELECT query specifies the columns to be retrieved as a column list. The syntax for a basic SELECT query that retrieves data from a table is:

SELECT column-list

FROM table-list;

The column-list represents one or more attributes, separated by commas. If the programmer wants all the columns to be returned, then the asterisk (*) wildcard can be used. A **wildcard character** is a symbol that can be used as a general substitute for other characters or commands. This wildcard means "all columns." For example, the following query would return all the data from the PRODUCT table.

SELECT * FROM PRODUCT;

In this query, the column list indicates that all columns (and by default all of the rows) should be returned. The FROM clause specifies that the data from the PRODUCT table is to be used. To limit the rows being returned, relational selection (or restriction) must be used. The column list allows the programmer to specify which columns should be returned, as shown in the next query.

SELECT P_CODE, P_DESCRIPT, P_PRICE, P_QOH

FROM PRODUCT;

This query specifies that the data should come from the PRODUCT table, and that only the product code, description, price, and quantity on hand columns should be included. Notice that only the requested columns are returned and that the columns are in the same order in the output as they were listed in the query.

Using Computed Columns

A computed column (also called a calculated column) represents a derived attribute. For example, suppose that you want to determine the total value of each of the products currently held in inventory. You can accomplish this task with the following command:

SELECT *P_DESCRIPT, P_QOH, P_PRICE,* ***P_QOH * P_PRICE***

FROM *PRODUCT*;

SQL accepts any valid expressions (or formulas) in the computed columns. Such formulas can contain any valid mathematical operators and functions that are applied to attributes in any of the tables specified in the FROM clause of the SELECT statement. Different DBMS products vary in the column headings that are displayed for the computed column.

To make the output more readable, an alias is typically used for any computed fields. An **alias**, in simple terms, is a column name that is defined by the programmer within an SQL query. For example, you can rewrite the previous SQL statement as follows:

SELECT *P_DESCRIPT, P_QOH, P_PRICE, P_QOH * P_PRICE* ***AS*** *TOTVALUE*

FROM *PRODUCT*;

Arithmetic Operators: The Rule of Precedence

SQL commands are often used in conjunction with the arithmetic operators shown in Table 7.4.

TABLE 7.4

THE ARITHMETIC OPERATORS

OPERATOR	DESCRIPTION
+	Add
-	Subtract
*	Multiply
/	Divide
^	Raise to the power of (some applications use ** instead of ^)

As you perform mathematical operations on attributes, remember the mathematical rules of precedence. As the name suggests, the rules of precedence are the rules that establish the order in which computations are completed. For example, note the order of the following computational sequence:

1. Perform operations within parentheses.
2. Perform power operations.
3. Perform multiplications and divisions.
4. Perform additions and subtractions.

Date Arithmetic

Date data in the column list can be interesting when used in computed fields. Internally, the DBMS stores a date value in a numeric format. Although the details can be complicated, essentially, a date is stored as a day number, that is, the number of days that have passed since some defined point in history. Exactly what that point in history is varies from one DBMS to another. However, because the values are stored as a number of days, it is possible to perform date arithmetic in a query. For example, if today's date in some DBMS is the day number "250,000," then tomorrow will be "250,001," and yesterday was "249,999." Adding or subtracting a number from a date that is stored in a date data type returns the date that is the specified number of days from the given date. Subtracting one date value from another yields the number of days between those dates. Suppose that a manager wants a list of all products, the dates they were received, and the warranty expiration date (90 days from receiving the product). To generate that list, you would make the following query:

SELECT *P_CODE, P_INDATE,* ***P_INDATE + 90 AS EXPDATE***

FROM *PRODUCT*;

The DBMS also has a function to return the current date on the database server, making it possible to write queries that reference the current date without having to change the contents of the query each day. For example, the DATE(), GETDATE(), and CURDATE() functions in MS Access, SQL Server, and MySQL, respectively, and the SYSDATE keyword in Oracle will all retrieve the current date.

Listing Unique Values

SQL's **DISTINCT** clause produces a list of only those values that are different from one another. For example, the command

SELECT **DISTINCT** *V_CODE*

FROM *PRODUCT*;

yields only the different vendor codes (V_CODE) in the PRODUCT table. The DISTINCT keyword only appears once in the query, and that is immediately following the SELECT keyword.

Read

Database Systems: Design, Implementation, and Management, 13th Edition Carlos Coronel and Steven Morris, Chapter Seven, page 250-256.

FROM Clause Options

The **FROM** clause of the query specifies the table or tables from which the data is to be retrieved. In practice, most SELECT queries will need to retrieve data from multiple tables. Through joins, the programmer integrates pieces of data to satisfy the users' information needs.

Inner joins return only rows from the tables that match on a common value. *Outer joins* return the same matched rows as the inner join, plus unmatched rows from one table or the other.

The join condition is generally composed of an equality comparison between the foreign key and the primary key of related tables. For example, suppose that you want to join the two tables VENDOR and PRODUCT. Because V_CODE is the foreign key in the PRODUCT table and the primary key in the VENDOR table, the link is established on V_CODE.

Joining the PRODUCT and VENDOR tables, can be accomplished in multiple ways.

Natural Join

Recall that a natural join returns all rows with matching values in the matching columns and eliminates duplicate columns. This style of query is used when the tables share one or more common attributes with common names. The natural join syntax is:

SELECT *column-list*

FROM *table1*

NATURAL JOIN *table2*

The natural join performs the following tasks:

- Determines the common attribute(s) by looking for attributes with identical names and compatible data types.
- Selects only the rows with common values in the common attribute(s).
- If there are no common attributes, returns the relational product of the two tables.

The following example performs a natural join of the CUSTOMER and INVOICE tables and returns only selected attributes:

SELECT *CUS_CODE, CUS_LNAME, INV_NUMBER, INV_DATE*

FROM *CUSTOMER* **NATURAL JOIN** *INVOICE*;

You are not limited to two tables when performing a natural join.

JOIN USING Syntax

A second way to express a join is through the USING keyword. The query returns only the rows with matching values in the column indicated in the USING clause—and that column must exist in both tables. The syntax is:

SELECT *column-list*

FROM *table1*

JOIN *table2* USING (common-column)

To see the JOIN USING query in action, perform a join of the INVOICE and LINE tables by writing the following:

SELECT *P_CODE, P_DESCRIPT, V_CODE, V_NAME, V_AREACODE, V_PHONE*

FROM *PRODUCT*

JOIN *VENDOR* **USING** (*V_CODE*);

The preceding SQL command sequence joins a row in the PRODUCT table with a row in the VENDOR table, in which the V_CODE values of these rows are the same, as indicated in the USING clause. As with the NATURAL JOIN command, the JOIN USING operand does not require table qualifiers and only returns one copy of the common attribute.

JOIN ON Syntax

The previous two join styles use common attribute names in the joining tables. Another way to express a join when the tables have no common attribute names is to use the JOIN ON operand. The query returns only the rows that meet the indicated join condition. The join condition typically includes an equality comparison expression of two columns. (The columns may or may not share the same name, but they must have comparable data types.) The syntax is:

SELECT *column-list*

FROM *table1*

JOIN *table2* ON join-condition

The following example performs a join of the INVOICE and LINE tables using the ON clause.

SELECT *INVOICE.INV_NUMBER, PRODUCT.P_CODE, P_DESCRIPT, LINE_UNITS, LINE_PRICE*

FROM *INVOICE JOIN LINE ON INVOICE.INV_NUMBER = LINE.INV_NUMBER*

JOIN *PRODUCT* **ON** *LINE.P_CODE = PRODUCT.P_CODE*;

Outer Joins

An outer join returns not only the rows matching the join condition (that is, rows with matching values in the common columns), but it also returns the rows with unmatched values. The ANSI standard defines three types of outer joins: left, right, and full. The left and right designations reflect the order in which the tables are processed by the DBMS. Remember that join operations take place two tables at a time. The first table named in the FROM clause will be the left side, and the second table named will be the right side. If three or more tables are being joined, the result of joining the first two tables becomes the left side, and the third table becomes the right side. The left outer join returns not only the rows matching the join condition (that is, rows with matching values in the common column), but it also returns the rows in the left table with unmatched values in the right table. The syntax is:

SELECT *column-list*

FROM *table1*

LEFT [OUTER] JOIN *table2* ON *join-condition*

For example, the following query lists the product code, vendor code, and vendor name for all products and includes those vendors with no matching products:

SELECT *P_CODE, VENDOR.V_CODE, V_NAME*

FROM *VENDOR*

LEFT JOIN *PRODUCT* ON *VENDOR. V_CODE = PRODUCT.V_CODE*;

The right outer join returns not only the rows matching the join condition (that is, rows with matching values in the common column), but it also returns the rows in the right table with unmatched values in the left table.

The full outer join returns not only the rows matching the join condition (that is, rows with matching values in the common column), but it also returns all the rows with unmatched values in the table on either side. The syntax is like that of the left and full join; you simply need to replace the LEFT with RIGHT or FULL.

In addition to the JOIN operations discussed above, other joins, such as cross joins and recursive joins, may be performed.

A **cross join** performs a relational product (also known as the Cartesian product) of two tables.

A **recursive join** is a join performed when a table must be joined to itself, as is the case when working with unary relationships.

Read

Database Systems: Design, Implementation, and Management, 13th Edition Carlos Coronel and Steven Morris, Chapter Seven, page 256-266.

ORDER BY Clause Options

The **ORDER BY** clause is especially useful when the listing order is important to you. The syntax is:

SELECT *column-list*

FROM *table-list*

[ORDER BY *column-list* [ASC | DESC];

Although you have the option of declaring the order type—ascending or descending—the default order is ascending. For example, if you want the contents of the PRODUCT table to be listed by P_PRICE in ascending order, use the following command:

```
SELECT      P_CODE, P_DESCRIPT, P_QOH, P_PRICE
FROM        PRODUCT
ORDER BY    P_PRICE;
```

Note that although ORDER BY produces a sorted output, the actual table contents are unaffected by the ORDER BY operation.

You can add DESC after the attribute to indicate descending order. To produce the listing with products sorted in descending order by price, you would enter:

```
SELECT      P_CODE, P_DESCRIPT, P_QOH, P_PRICE
FROM        PRODUCT
ORDER BY    P_PRICE DESC;
```

Ordered listings are used frequently and we often order data in multilevel ordered sequences (by more than one attribute). A multilevel ordered sequence is known as a **cascading order sequence**, and it can be created by listing several attributes, separated by commas, after the ORDER BY clause.

To illustrate a cascading order sequence, use the following SQL command on the EMPLOYEE table:

```
SELECT      EMP_LNAME, EMP_FNAME, EMP_INITIAL, EMP_AREACODE,
            EMP_PHONE
FROM        EMPLOYEE
ORDER BY    EMP_LNAME, EMP_FNAME, EMP_INITIAL;
```

Read

Database Systems: Design, Implementation, and Management, 13th Edition Carlos Coronel and Steven Morris, Chapter Seven, page 266-269.

WHERE Clause Options

In this section, you learn how to fine-tune the SELECT command by adding restrictions to the search criteria. When coupled with appropriate search conditions, SELECT is an incredibly powerful tool that enables you to transform data into information.

Selecting Rows with Conditional Restrictions

You can select partial table contents by placing restrictions on the rows to be included in the output. Use the **WHERE** clause to add conditional restrictions to the SELECT statement that limit the rows returned by the query. The following syntax enables you to specify which rows to select:

SELECT *column-list*

FROM *table-list*

[WHERE *condition-list*]

[ORDER BY *column-list* [ASC | DESC];

The SELECT statement retrieves all rows that match the specified condition(s)—also known as the conditional criteria—you specified in the WHERE clause. The condition-list in the WHERE clause of the SELECT statement is represented by one or more conditional expressions, separated by logical operators. The WHERE clause is optional. If no rows match the specified criteria in the WHERE clause, you see a blank screen or a message that tells you no rows were retrieved. For example, consider the following query:

SELECT *P_DESCRIPT, P_QOH, P_PRICE, V_CODE*

FROM *PRODUCT*

WHERE *V_CODE = 21344*;

This query returns the description, quantity on hand, price, and vendor code for products with a vendor code of 21344, as shown in Figure 7.21.

FIGURE 7.21 SELECTED PRODUCT ATTRIBUTES FOR VENDOR CODE 21344

P_DESCRIPT	P_QOH	P_PRICE	V_CODE
7.25-in. pwr. saw blade	32	14.99	21344
9.00-in. pwr. saw blade	18	17.49	21344
Rat-tail file, 1/8-in. fine	43	4.99	21344

Numerous conditional restrictions can be placed on the selected table contents. For example, the comparison operators shown in Table 7.6 can be used to restrict output. Note that there are two options for not equal to. Both <> and != are well supported and perform the same function.

TABLE 7.6

COMPARISON OPERATORS

SYMBOL	MEANING
=	Equal to
<	Less than
<=	Less than or equal to
>	Greater than
>=	Greater than or equal to
<> or !=	Not equal to

Using Comparison Operators on Character Attributes

Because computers identify all characters by their numeric American Standard Code for Information Interchange (ASCII) codes, comparison operators may even be used to place restrictions on character-based attributes. Therefore, the command:

SELECT *P_CODE, P_DESCRIPT, P_QOH, P_MIN, P_PRICE*

FROM *PRODUCT*

WHERE *P_CODE < '1558-QW1'*;

would be correct and would yield a list of all rows in which the P_CODE is alphabetically less than 1558-QW1.

String (character) comparisons are made from left to right. This left-to-right comparison is especially useful when attributes such as names are to be compared. However, may get some unexpected results from comparisons when dates or other numbers are stored in character format.

Using Comparison Operators on Dates

Date procedures are often more software-specific than other SQL procedures. For example, the query to list all the rows in which the inventory stock dates occur on or after January 20, 2018, may look like this:

SELECT *P_DESCRIPT, P_QOH, P_MIN, P_PRICE, P_INDATE*

FROM *PRODUCT*

WHERE *P_INDATE >= '20-Jan-2018'*;

The date value format is dependent on the RDMBS that the programmer is using. For example, in MySQL, the expected date format is yyyy-mm-dd, so the WHERE clause would be written as:

WHERE *P_INDATE >= '2018-01-20'*

Logical Operators: AND, OR, and NOT

SQL allows you to include multiple conditions in a query through logical operators. The logical operators are AND, OR, and NOT. For example, if you want a list of the table contents for either the V_CODE = 21344 or the V_CODE = 24288, you can use the **OR** logical operator, as in the following command sequence:

SELECT *P_DESCRIPT, P_QOH, P_PRICE, V_CODE*

FROM *PRODUCT*

WHERE *V_CODE = 21344* **OR** *V_CODE = 24288*;

The logical operator **AND** has the same SQL syntax requirement as OR. Furthermore, you can combine the logical OR with the logical AND to place further restrictions on the output, with the use of parenthesis to group clauses appropriately.

The use of the logical operators OR and AND can become quite complex when numerous restrictions are placed on the query. In fact, a specialty field in mathematics known as **Boolean algebra** is dedicated to the use of logical operators.

The logical operator **NOT** is used to negate the result of a conditional expression. That is, in SQL, all conditional expressions evaluate to true or false. If an expression is true, the row is selected; if

an expression is false, the row is not selected. The NOT logical operator is typically used to find the rows that do not match a certain condition. For example, if you want to see a listing of all rows for which the vendor code is not 21344, use the following command sequence:

```
SELECT    *
FROM      PRODUCT
WHERE     NOT    (V_CODE = 21344);
```

Note that the condition is enclosed in parentheses; that practice is optional, but it is highly recommended for clarity. The logical operator NOT can be combined with AND and OR.

Read

Database Systems: Design, Implementation, and Management, 13th Edition Carlos Coronel and Steven Morris, Chapter Seven, page 269-275.

Aggregate Processing

The RDBMS works on sets of data, not individual row processing, but one can imagine the processing that has been discussed up to this point as if it were row-based. However, there are many questions that are asked of the database that require working with collections of rows as if they are a single unit. This type of collection processing is done with aggregate functions. The defining characteristic of using an aggregate function is that it takes a collection of rows and reduces it to a single row. SQL provides useful aggregate functions that count, find minimum and maximum values, calculate averages, and so on. Better yet, SQL allows the user to limit queries to only those entries that have no duplicates or entries whose duplicates can be grouped.

Aggregate Functions

SQL can perform various mathematical summaries for you, such as counting the number of rows that contain a specified condition, finding the minimum or maximum values for a specified attribute, summing the values in a specified column, and averaging the values in a specified column. While there are other aggregate functions supported by some DBMS products, the ones shown in Table 7.7 are the most common aggregate functions and are supported by most DBMS products. Aggregate functions are most commonly used in the SELECT column list to return an aggregate value that has been calculated across a collection of rows and take a value, typically an attribute, as a parameter inside parentheses.

TABLE 7.7

SOME BASIC SQL AGGREGATE FUNCTIONS

FUNCTION	OUTPUT
COUNT	The number of rows containing non-null values
MIN	The minimum attribute value encountered in a given column
MAX	The maximum attribute value encountered in a given column
SUM	The sum of all values for a given column
AVG	The arithmetic mean (average) for a specified column

COUNT The **COUNT** function is used to tally the number of non-null values of an attribute. In the following code, a tally of the number of products is calculated, returning a result of 16.

```
SELECT    COUNT (P_CODE)
FROM      PRODUCT;
```

The collection of rows does not have to be composed of all the rows in the table. For example, we can use the following query to determine how many products have a price that is less than $10.

```
SELECT    COUNT(P_PRICE)
FROM      PRODUCT
WHERE     P_PRICE < 10;
```

COUNT can also be used in conjunction with the DISTINCT clause.

MIN and MAX The **MIN** and **MAX** functions help you find answers to problems such as the highest and lowest (maximum and minimum) prices in the PRODUCT table. The following code retrieves the highest and lowest prices in the PRODUCT table in a single query.

```
SELECT    MAX(P_PRICE) AS MAXPRICE, MIN(P_PRICE) AS MINPRICE
FROM      PRODUCT;
```

SUM and AVG The **SUM** function computes the total sum for any specified numeric attribute, using any condition(s) you have imposed. For example, if you want to compute the total amount owed by your customers, you could use the following command:

```
SELECT    SUM(CUS_BALANCE) AS TOTBALANCE
FROM      CUSTOMER;
```

The **AVG** function format is similar to those of SUM and is subject to the same operating restrictions. The following command set shows how a simple average P_PRICE value can be generated to yield the computed average price of 56.42125.

```
SELECT    AVG(P_PRICE) AS AVGPRICE
FROM      PRODUCT;
```

Grouping Data

Sometimes, you do not want to treat the entire table as a single collection of data for summarizing. Rows can be grouped into smaller collections quickly and easily using the **GROUP BY** clause within the SELECT statement. The aggregate functions will then summarize the data within each smaller collection. The syntax is:

SELECT *column-list*

FROM *table-list*

[WHERE *condition-list*]

[GROUP BY *column-list*]

[ORDER BY *column-list* [ASC | DESC]];

What if instead of seeing the price across all products, the users wanted to see the average price of the products provided by each vendor? The following query will answer that question.

SELECT *V_CODE, AVG(P_PRICE) AS AVGPRICE*

FROM *PRODUCT*

GROUP BY *V_CODE*;

Having Clause

Aggregate functions are very powerful and are used frequently in reporting. However, restricting data based on an aggregate value is slightly more complicated and can require the use of a HAVING clause. The syntax for a HAVING clause is:

SELECT *column-list*

FROM *table-list*

[WHERE *condition-list*]

[GROUP BY *column-list*]

[HAVING *condition-list*]

[ORDER BY *column-list* [ASC | DESC]];

The HAVING clause operates very much like the WHERE clause in the SELECT statement. However, the WHERE clause applies to columns and expressions for individual rows, whereas the HAVING clause is applied to the output of a GROUP BY operation. For example, suppose that you want to generate a listing of the number of products in the inventory supplied by each vendor. However, this time you want to limit the listing to products whose prices average less than $10. The query requires both a GROUP BY clause and a HAVING clause.

SELECT *V_CODE, COUNT(P_CODE) AS NUMPRODS*

FROM *PRODUCT*

GROUP *BY V_CODE*

HAVING *AVG(P_PRICE) < 10*

ORDER BY *V_CODE*;

Read

Database Systems: Design, Implementation, and Management, 13th Edition Carlos Coronel and Steven Morris, Chapter Seven, page 281-290.

Subqueries

The use of joins in a relational database allows you to get information from two or more tables. However, it is often necessary to process data based on other processed data. For example, suppose that you want to generate a list of vendors who do not provide products. This result can also be found by using a subquery, such as:

SELECT *V_CODE, V_NAME*

FROM *VENDOR*

WHERE *V_CODE* NOT IN **(SELECT V_CODE FROM PRODUCT WHERE V_CODE IS NOT NULL);**

Similarly, to generate a list of all products with a price greater than or equal to the average product price, you can write the following query:

SELECT *P_CODE, P_PRICE*

FROM *PRODUCT*

WHERE *P_PRICE* >= **(SELECT AVG(P_PRICE) FROM PRODUCT)**;

Likewise, to list all products with a total quantity sold greater than the average quantity sold, you would write the following query:

SELECT *P_CODE, SUM(LINE_UNITS) AS TOTALUNITS*

FROM *LINE*

GROUP BY *P_CODE*

HAVING *SUM(LINE_UNITS)* > **(SELECT AVG(LINE_UNITS) FROM LINE)**;

In both queries, you needed to get information that was not unknown, and you used a subquery to generate the required information, which could then be used as input for the originating query. There are key characteristics that you should remember for subqueries:

- A subquery is a query (SELECT statement) inside another query.
- A subquery is normally expressed inside parentheses.
- The first query in the SQL statement is known as the *outer query*
- The query inside the SQL statement is known as the *inner query*.
- The inner query is executed first.
- The output of an inner query is used as the input for the outer query.
- The entire SQL statement is sometimes referred to as a *nested query*.

The subquery is always on the right side of a comparison or assigning expression. Also, a subquery can return one or more values. To be precise, the subquery can return the following:

- One single value (one column and one row).
- A list of values (one column and multiple rows).
- A virtual table (multicolumn, multirow set of values).

Multirow Subquery Operators: ALL and ANY

So far, you have learned that you must use an IN subquery to compare a value to a list of values. However, the IN subquery uses an equality operator; that is, it selects only those rows that are equal to at least one of the values in the list. What happens if you need to make an inequality comparison (> or <) of one value to a list of values?

For example, suppose you want to know which products cost more than all individual products provided by vendors from Florida:

```
SELECT   P_CODE, P_QOH * P_PRICE AS TOTALVALUE
FROM     PRODUCT
WHERE    P_QOH * P_PRICE > ALL (SELECT P_QOH * P_PRICE FROM PRODUCT
         WHERE V_CODE IN (SELECT V_CODE FROM VENDOR WHERE V_STATE =
         'FL'));
```

Read

Database Systems: Design, Implementation, and Management, 13th Edition Carlos Coronel and Steven Morris, Chapter Seven, page 290-302.

SQL Functions

The data in databases is the basis of critical business information. Generating information from data often requires many data manipulations. Sometimes such data manipulation involves the decomposition of data elements.

SQL functions are very useful tools. Functions always use a numerical, date, or string value. The value may be part of the command itself (a constant or literal), or it may be an attribute located in a table. Therefore, a function may appear anywhere in a SQL statement where a value or an attribute can be used.

There are many types of SQL functions, such as arithmetic, trigonometric, string, date, and time functions. See prescribed textbook for further details.

Read

Database Systems: Design, Implementation, and Management, 13th Edition Carlos Coronel and Steven Morris, Chapter Seven, page 302-311.

Relational Set Operators

In this section, you will learn how to use three SQL operators—UNION, INTERSECT, and EXCEPT (MINUS)—to implement the union, intersection, and difference relational operators. You can combine two or more sets to create new sets (or relations). That is precisely what the UNION, INTERSECT, and EXCEPT (MINUS) statements do.

UNION, INTERSECT, and EXCEPT (MINUS) work properly only if relations are *union-compatible*, which means that the number of attributes must be the same and their corresponding data types must be alike.

UNION

Suppose that SaleCo has bought another company. SaleCo's management wants to make sure that the acquired company's customer list is properly merged with its own customer list. Because some customers might have purchased goods from both companies, the two lists might contain common customers. SaleCo's management wants to make sure that customer records are not duplicated when the two customer lists are merged. The UNION query is a perfect tool for generating a combined listing of customers—one that excludes duplicate records.

The UNION statement combines rows from two or more queries without including duplicate rows. The syntax of the UNION statement is:

query **UNION** *query*

In other words, the UNION statement combines the output of two SELECT queries. To demonstrate the use of the UNION statement in SQL, use the CUSTOMER and CUSTOMER_2 tables in the Ch07_SaleCo database. To show the combined CUSTOMER and CUSTOMER_2 records without duplicates, the UNION query is written as follows:

SELECT *CUS_LNAME, CUS_FNAME, CUS_INITIAL, CUS_AREACODE, CUS_PHONE*

FROM *CUSTOMER*

UNION

SELECT *CUS_LNAME, CUS_FNAME, CUS_INITIAL, CUS_AREACODE, CUS_PHONE*

FROM *CUSTOMER_2*;

The UNION statement can be used to unite more than just two queries. For example, assume that you have four union-compatible queries named T1, T2, T3, and T4. With the UNION statement, you can combine the output of all four queries into a single result set. The SQL statement will be similar to this:

SELECT *column-list* FROM *T1*

UNION

SELECT *column-list* FROM *T2*

UNION

SELECT *column-list* FROM *T3*

UNION

SELECT *column-list* FROM *T4*;

UNION ALL

If SaleCo's management wants to know how many customers are on both the CUSTOMER and CUSTOMER_2 lists, a UNION ALL query can be used to produce a relation that retains the duplicate rows. The syntax is the same as that of the UNION query, where the UNION is replaced by the UNION ALL clause.

INTERSECT

the INTERSECT statement can be used to combine rows from two queries, returning only the rows that appear in both sets. The syntax for the INTERSECT statement is:

query **INTERSECT** *query*

The INTERSECT statement can be used to generate additional useful customer information. For example, the following query returns the customer codes for all customers who are in area code 615 and who have made purchases. (If a customer has made a purchase, there must be an invoice record for that customer.)

SELECT *CUS_CODE*

FROM *CUSTOMER*

WHERE *CUS_AREACODE = '615'*

INTERSECT

SELECT *DISTINCT CUS_CODE*

FROM *INVOICE*;

EXCEPT (MINUS)

The EXCEPT statement in SQL combines rows from two queries and returns only the rows that appear in the first set but not in the second. The syntax for the EXCEPT statement in MS SQL Server and the MINUS statement in Oracle is:

query EXCEPT query

and

query MINUS query

For example, if the SaleCo managers want to know which customers in the CUSTOMER table are not found in the CUSTOMER_2 table, they can use the following command in Oracle (see Figure 7.62).

SELECT *CUS_LNAME, CUS_FNAME, CUS_INITIAL, CUS_AREACODE, CUS_PHONE*

FROM *CUSTOMER*

MINUS

SELECT *CUS_LNAME, CUS_FNAME, CUS_INITIAL, CUS_AREACODE, CUS_PHONE*

FROM *CUSTOMER_2*;

Users of MS SQL Server would substitute the keyword EXCEPT in place of MINUS, but otherwise the syntax is the same. You can extract useful information by combining MINUS with various clauses such as WHERE.

Read

Database Systems: Design, Implementation, and Management, 13th Edition Carlos Coronel and Steven Morris, Chapter Seven, page 311-315.

4.2 Advanced SQL

Data Definition Commands

The same simple database composed of CUSTOMER, INVOICE, LINE, PRODUCT, and VENDOR tables that was used in the previous sections of this chapter is used as a starting point for illustrating the SQL commands in this section and subsequent sections.

Creating the Database

Before you can use a new RDBMS, you must complete two tasks: create the database structure and create the tables that will hold the end-user data. To complete the first task, the RDBMS creates the physical files that will hold the database. When you create a new database, the RDBMS automatically creates the data dictionary tables in which to store the metadata and creates a default database administrator.

If you work in a database environment typically used by larger organizations, you will probably use an enterprise RDBMS such as Oracle, MS SQL Server, MySQL, or DB2. For the purpose of this course we will be using the MySQL RDBMS and/or Microsoft Access.

If you are using an enterprise RDBMS, you must be authenticated by the RDBMS before you can start creating tables. **Authentication** is the process the DBMS uses to verify that only registered users access the database. To be authenticated, you must log on to the RDBMS using a user ID and a password created by the database administrator. In an enterprise RDBMS, every user ID is associated with a database schema.

The Database Schema

In the SQL environment, a schema is a logical group of database objects—such as tables and indexes—that are related to each other. Usually, the schema belongs to a single user or application. A single database can hold multiple schemas that belong to different users or applications. Schemas are useful in that they group tables by owner (or function) and enforce a first level of security by allowing each user to see only the tables that belong to that user.

ANSI SQL standards define a command to create a database schema:

CREATE SCHEMA AUTHORIZATION {creator};

This command is seldom used directly and for most RDBMSs, the CREATE SCHEMA AUTHORIZATION command is optional.

Data Types

There is a wide array of data types supported by SQL. Only a few of the most common are covered in this book. Generally speaking, the data types are character, numeric, and date (see Table 8.1).

TABLE 8.1

SOME COMMON SQL DATA TYPES

DATA TYPE	FORMAT	COMMENTS
Numeric	NUMBER(L,D) or NUMERIC(L,D)	The declaration NUMBER(7,2) or NUMERIC(7,2) indicates that numbers will be stored with two decimal places and may be up to seven digits long, including the sign and the decimal place (for example, 12.32 or –134.99).
	INTEGER	May be abbreviated as INT. Integers are (whole) counting numbers, so they cannot be used if you want to store numbers that require decimal places.
	SMALLINT	Like INTEGER but limited to integer values up to six digits. If your integer values are relatively small, use SMALLINT instead of INT.
	DECIMAL(L,D)	Like the NUMBER specification, but the storage length is a *minimum* specification. That is, greater lengths are acceptable, but smaller ones are not. DECIMAL(9,2), DECIMAL(9), and DECIMAL are all acceptable.
Character	CHAR(L)	Fixed-length character data for up to 255 characters. If you store strings that are not as long as the CHAR parameter value, the remaining spaces are left unused. Therefore, if you specify CHAR(25), strings such as *Smith* and *Katzenjammer* are each stored as 25 characters. However, a U.S. area code is always three digits long, so CHAR(3) would be appropriate if you wanted to store such codes.
	VARCHAR(L) or VARCHAR2(L)	Variable-length character data. The designation VARCHAR2(25) or VARCHAR(25) will let you store characters up to 25 characters long. However, unlike CHAR, VARCHAR will not leave unused spaces. Oracle automatically converts VARCHAR to VARCHAR2.
Date	DATE	Stores dates in the Julian date format.

Note that if your RDBMS is fully compliant with ANSI SQL, it supports many more data types than those shown in Table 8.1. Those data types, include TIME, TIMESTAMP, REAL, DOUBLE, and FLOAT, and intervals, such as INTERVAL DAY TO HOUR. However, for the purpose of this chapter, the discussion is limited to the data types summarized in Table 8.1.

Read

Database Systems: Design, Implementation, and Management, 13th Edition Carlos Coronel and Steven Morris, Chapter Eight, page 360-366.

Creating Table Structures

Once the database has been created and the appropriate data types for each attribute have been determined, it is time to create the actual database tables. Recall that when implementing the database design, every entity becomes a table, and the attributes of each entity become the columns in that table.

Now you are ready to implement the PRODUCT and VENDOR table structures with the help of SQL, using the **CREATE TABLE** syntax shown next.

CREATE TABLE table-name (

column1 *data type* [*constraint*] [,

column2 *data type* [*constraint*] [,

PRIMARY KEY (*column1* [, *column2*])] [,

FOREIGN KEY (*column1* [, *column2*]) REFERENCES *table-name*] [,

CONSTRAINT *constraint*]);

The following examples that create VENDOR and PRODUCT tables, illustrate the CREATE TABLE command.

Example 1

CREATE TABLE VENDOR (

V_CODE	INTEGER	NOT NULL	UNIQUE,
V_NAME	VARCHAR(35)	NOT NULL,	
V_CONTACT	VARCHAR(25)	NOT NULL,	
V_AREACODE	CHAR(3)	NOT NULL,	
V_PHONE	CHAR(8)	NOT NULL,	
V_STATE	CHAR(2)	NOT NULL,	
V_ORDER	CHAR(1)	NOT NULL,	

PRIMARY KEY (V_CODE));

Example 2

CREATE TABLE PRODUCT (

P_CODE	VARCHAR(10)	NOT NULL	UNIQUE,
P_DESCRIPT	VARCHAR(35)	NOT NULL,	
P_INDATE	DATE	NOT NULL,	
P_QOH	SMALLINT	NOT NULL,	
P_MIN	SMALLINT	NOT NULL,	
P_PRICE	NUMBER(8,2)	NOT NULL,	
P_DISCOUNT	NUMBER(5,2)	NOT NULL,	

V_CODE INTEGER,

PRIMARY KEY (P_CODE),

FOREIGN KEY (V_CODE) REFERENCES VENDOR ON UPDATE CASCADE);

As you examine the preceding SQL table-creating command sequences, note the following features:

- The NOT NULL specifications for the attributes ensure that a data entry will be made. When it is crucial to have the data available, the NOT NULL specification will not allow the end user to leave the attribute empty (with no data entry at all).
- The UNIQUE specification creates a unique index on the respective attribute. Use it to avoid having duplicated values in a column.
- The primary key attributes contain both a NOT NULL and UNIQUE specification, which enforce the entity integrity requirements.
- The entire table definition is enclosed in parentheses. A comma is used to separate each table element definition (attributes, primary key, and foreign key).
- The ON UPDATE CASCADE specification ensures that if you make a change in any VENDOR's V_CODE that change is automatically applied to all foreign key references throughout the system to ensure that referential integrity is maintained. That restriction makes it impossible for a V_CODE value to exist in the PRODUCT table if it points to a non-existent VENDOR table V_CODE value.
- An RDBMS automatically enforces referential integrity for foreign keys. That is, you cannot have an invalid entry in the foreign key column; at the same time, you cannot delete a vendor row as long as a product row references that vendor.
- The command sequence ends with a semicolon.

SQL Constraints

You learned that adherence to rules for entity integrity and referential integrity is crucial in a relational database environment. Fortunately, most SQL implementations support both integrity rules. Entity integrity is enforced automatically when the primary key is specified in the CREATE TABLE command sequence.

In the PRODUCT table's CREATE TABLE sequence, note that referential integrity has been enforced by specifying the following in the PRODUCT table:

FOREIGN KEY (V_CODE) REFERENCES VENDOR ON UPDATE CASCADE

In general, ANSI SQL permits the use of ON DELETE and ON UPDATE clauses to cover CASCADE, SET NULL, or SET DEFAULT. Besides the PRIMARY KEY and FOREIGN KEY constraints, the ANSI SQL standard also defines the following constraints:

- The DEFAULT constraint assigns an 'automatic' value to an attribute when a new row is added to a table.
- The CHECK constraint is used to validate data when an attribute value is entered. The CHECK constraint does precisely what its name suggests: it checks to see that a specified condition exists.

The following SQL command sequence uses the DEFAULT and CHECK constraints to define the table named CUSTOMER.

```
CREATE TABLE CUSTOMER (
CUS_CODE        NUMBER          PRIMARY KEY,
CUS_LNAME       VARCHAR(15)     NOT NULL,
CUS_FNAME       VARCHAR(15)     NOT NULL,
CUS_INITIAL     CHAR(1),
CUS_AREACODE CHAR(3)            DEFAULT '615'       NOT NULL
                                CHECK(CUS_AREACODE  IN
                                ('615','713','931')),
CUS_PHONE       CHAR(8)         NOT NULL,
CUS_BALANCE     NUMBER(9,2)     DEFAULT 0.00,
CONSTRAINT CUS_UI1 UNIQUE (CUS_LNAME, CUS_FNAME));
```

The final SQL command sequence creates the LINE table. The LINE table has a composite primary key (INV_NUMBER, LINE_NUMBER) and uses a UNIQUE constraint in INV_NUMBER and P_CODE to ensure that the same product is not ordered twice in the same invoice.

```
CREATE TABLE LINE (
INV_NUMBER      NUMBER          NOT NULL,
LINE_NUMBER     NUMBER(2,0)     NOT NULL,
P_CODE          VARCHAR(10)     NOT NULL,
LINE_UNITS      NUMBER(9,2)     DEFAULT 0.00        NOT NULL,
LINE_PRICE      NUMBER(9,2)     DEFAULT 0.00        NOT NULL,
PRIMARY KEY (INV_NUMBER, LINE_NUMBER),
FOREIGN KEY (INV_NUMBER) REFERENCES INVOICE ON DELETE CASCADE,
FOREIGN KEY (P_CODE) REFERENCES PRODUCT(P_CODE),
CONSTRAINT LINE_UI1 UNIQUE(INV_NUMBER, P_CODE));
```

In the creation of the LINE table, note that a UNIQUE constraint is added to pre-vent the duplication of an invoice line. A UNIQUE constraint is enforced through the creation of a unique index.

Create a Table with a SELECT Statement

SQL provides a way to rapidly create a new table based on selected columns and rows of an existing table using a subquery. In this case, the new table copies the attribute names, data characteristics, and rows of the original table as retrieved by the subquery. The command is:

```
CREATE TABLE PART AS
SELECT    P_CODE AS PART_CODE, P_DESCRIPT AS PART_DESCRIPT,
          P_PRICE AS PART_PRICE, V_CODE
```

FROM PRODUCT;

The MS Access version of this command is:

```
SELECT    P_CODE AS PART_CODE, P_DESCRIPT AS PART_DESCRIPT, P_PRICE AS
          PART_PRICE, V_CODE INTO PART
FROM      PRODUCT;
```

However, note that no entity integrity (primary key) or referential integrity (foreign key) rules are automatically applied to the new table.

SQL Indexes

You've learned that indexes can be used to improve the efficiency of searches and to avoid duplicate column values. The ability to create indexes quickly and efficiently is important. Using the **CREATE INDEX** command, SQL indexes can be created on the basis of any selected attribute. The syntax is:

CREATE [UNIQUE]INDEX *index-name* ON *table-name*(*column1* [, *column2*])

For example, based on the attribute P_INDATE stored in the PRODUCT table, the following command creates an index named P_INDATEX:

CREATE INDEX P_INDATEX ON PRODUCT(P_INDATE);

Using the UNIQUE index qualifier, as shown in the command below, you can even create an index that prevents you from using a value that has been used before.

CREATE UNIQUE INDEX P_CODEX ON PRODUCT(P_CODE);

To delete an index, use the **DROP INDEX** command:

DROP INDEX *index-name*

After creating the tables and some indexes, you are ready to start entering data.

Read

Database Systems: Design, Implementation, and Management, 13th Edition Carlos Coronel and Steven Morris, Chapter Eight, page 366-375.

Altering Table Structures

In this section, you will learn how to change table structures by changing attribute characteristics and by adding columns. Finally, you will learn how to copy tables or parts of tables and how to delete tables.

All changes in the table structure are made by using the **ALTER TABLE** command followed by a keyword that produces the specific change you want to make. Three options are available: ADD, MODIFY, and DROP.

The basic syntax to add or modify columns is:

ALTER TABLE *table-name*

{ADD | MODIFY} (*column-name datatype* [{ADD | MODIFY} *column-name datatype*]);

The ALTER TABLE command can also be used to add table constraints. In those cases, the syntax would be:

ALTER TABLE *table-name* ADD *constraint* [ADD *constraint*];

You could also use the ALTER TABLE command to remove a column or table constraint. The syntax would be as follows:

ALTER TABLE *table-name* DROP {PRIMARY KEY | COLUMN *column-name* | CONSTRAINT *constraint-name*};

Changing a Column's Data Type and Characteristics

Using the ALTER syntax, the integer V_CODE in the PRODUCT table can be changed to a character V_CODE by using the following command:

ALTER TABLE PRODUCT

MODIFY (V_CODE CHAR(5));

Some RDBMSs do not let you change data types unless the column to be changed is empty.

If the column to be changed already contains data, you can make changes in the column's characteristics if those changes do not alter the data type. For example, if you want to increase the width of the P_PRICE column to nine digits, use the following command:

ALTER TABLE PRODUCT

MODIFY (P_PRICE DECIMAL(9,2));

Adding a Column

You can alter an existing table by adding one or more columns. In the following example, you add the column named P_SALECODE to the PRODUCT table, as follows:

ALTER TABLE PRODUCT

ADD (P_SALECODE CHAR(1));

When adding a column, be careful not to include the NOT NULL clause for the new column. Doing so will cause an error message; if you add a new column to a table that already has rows, the existing rows will default to a value of null for the new column.

Adding Primary Key, Foreign Key, and Check Constraints

When you create a new table based on another table, the new table does not include integrity rules from the old table. In particular, there is no primary key. To define the primary key, foreign key and check constraints for the new PART table, use the following commands:

Example 1

```
ALTER TABLE     PART
ADD                     PRIMARY KEY (PART_CODE);
```

Example 2

```
ALTER TABLE     PART
ADD             FOREIGN KEY (V_CODE) REFERENCES VENDOR;
```

Example 3

```
ALTER TABLE     PART
ADD             CHECK (PART_PRICE >= 0);
```

Note that these alterations can be made at once in a single command.

Dropping a Column

Occasionally, you might want to modify a table by deleting a column. Suppose that you want to delete the V_ORDER attribute from the VENDOR table. You would use the following command:

```
ALTER TABLE VENDOR
    DROP COLUMN V_ORDER;
```

Again, some RDBMSs impose restrictions on attribute deletion.

Deleting a Table from the Database

A table can be deleted from the database using the **DROP TABLE** command. For example, you can delete the PART table you just created with the following command:

```
DROP TABLE PART;
```

You can drop a table only if it is not the "one" side of any relationship. The order in which multiple tables must be dropped is influenced by the foreign key constraints. You should drop tables from the "many" side first, and then drop the table on the "one" side.

Read

Database Systems: Design, Implementation, and Management, 13th Edition Carlos Coronel and Steven Morris, Chapter Eight, page 375-378.

Data Manipulation Commands

In this section, you will learn how to use the basic SQL data manipulation commands INSERT, UPDATE, and DELETE, and the transaction management commands COMMIT and ROLLBACK.

Adding Table Rows

SQL requires the use of the insert command to enter data into a table. The INSERT command's basic syntax looks like this:

INSERT INTO *table-name* VALUES (*value1*, *value2*, ..., *value-n*)

Given the VENDOR table structure defined earlier you would enter the first two data rows as follows:

INSERT INTO VENDOR

VALUES (21225',Bryson, Inc.','Smithson','615','223-3234','TN','Y');

INSERT INTO VENDOR

VALUES (21226',Superloo, Inc.','Flushing','904','215-8995','FL','N');

Inserting Rows with Null Attributes In the previous example you entered rows in which all of the attribute values are specified. But what do you do if, for example, a product does not have a vendor or if you do not yet know the vendor code? In those cases, you would want to leave the vendor code null. To enter a null, use the following syntax:

INSERT INTO PRODUCT

VALUES ('BRT-345','Titanium drill bit','18-Oct-17', 75, 10, 4.50, 0.06, NULL);

Alternatively, to the syntax above, if more than one attribute is optional, you can indicate only the attributes that require values in the INSERT statement and omit the optional attributes. For the purpose of this example, assume that the only required attributes for the PRODUCT table are P_CODE and P_DESCRIPT:

INSERT INTO PRODUCT(P_CODE, P_DESCRIPT) VALUES ('BRT-345','Titanium drill bit');

Inserting Table Rows with a SELECT Subquery

Using a subquery with the INSERT command, it is possible to add multiple rows to a table, using another table as the source of the data, at the same time. The syntax is:

INSERT INTO *target-table-name*[(*target-column-list*)]

SELECT *source-column-list*

FROM *source-table-name*;

Note that the target column list is required if the source column list does not match all of the attribute names and characteristics of the target table (including the order of the columns).

For example, assume you already have the PART table and you want to copy data from the P_CODE, P_DESCRIPT, P_PRICE, and V_CODE columns of the PRODUCT table into it. You must specify the target column list in the following INSERT command because the column names of the target table are different:

INSERT INTO PART (PART_CODE, PART_DESCRIPT, PART_PRICE, V_CODE)
SELECT P_CODE, P_DESCRIPT, P_PRICE, V_CODE

FROM PRODUCT;

Given the previous SQL statement, the INSERT portion represents the outer query, and the SELECT portion represents the subquery.

The values returned by the SELECT subquery should match the attributes and data types of the table in the INSERT statement.

Saving Table Changes

Any changes made to the table contents are not saved on disk until you close the data-base, close the program you are using, or use the **COMMIT** command. The syntax for the COMMIT command is:

COMMIT [WORK]

The COMMIT command permanently saves all changes—such as rows added, attributes modified, and rows deleted—made to any table in the database.

However, the COMMIT command's purpose is not just to save changes. In fact, the ultimate purpose of the COMMIT and ROLLBACK commands is to ensure database update integrity in transaction management.

Updating Table Rows

Use the UPDATE command to modify data in a table. The syntax for this command is as follows:

UPDATE *table-name*

SET *column-name* = *expression* [, *column-name* = *expression*]

[WHERE *condition-list*];

For example, if you want to change P_INDATE from December 13, 2017, to January 18, 2018, in the second row of the PRODUCT table, use the primary key (13-Q2/P2) to locate the correct row. Therefore, type:

```
UPDATE    PRODUCT
SET       P_INDATE = '18-JAN-2018'
WHERE     P_CODE = '13-Q2/P2';
```

If you want to add 10 percent to the price for all products that have current prices below $50, you can use:

```
UPDATE    PRODUCT
SET       P_PRICE = P_PRICE * 1.10
WHERE     P_PRICE < 50.00;
```

Another example, if you want to enter the P_SALECODE value '1' for the P_CODE values '2232/QWE' and '2232/QTY', you use:

```
UPDATE    PRODUCT
SET       P_SALECODE = '1'
WHERE     P_CODE IN ('2232/QWE', '2232/QTY');
```

Deleting Table Rows

It is easy to delete a table row using the Delete statement. The syntax is:

DELETE FROM *table-name*

[WHERE *condition-list*];

For example, if you want to delete the product you added earlier whose code (P_CODE) is BRT-345, use the following command:

```
DELETE FROM    PRODUCT
WHERE          P_CODE = 'BRT-345';
```

Note that deletion is not limited to a primary key match; any attribute may be use. Likewise with updating.

Restoring Table Contents

If you have not yet used the COMMIT command to store the changes permanently in the database, you can restore the database to its previous condition with the **ROLLBACK** command. ROLLBACK undoes any changes since the last COMMIT command and brings all of the data back to the values that existed before the changes were made. To restore the data to its "pre-change" condition, type:

```
ROLLBACK;
```

and then press Enter.

COMMIT and ROLLBACK work only with data manipulation commands that add, modify, or delete table rows.

Read

Database Systems: Design, Implementation, and Management, 13th Edition Carlos Coronel and Steven Morris, Chapter Eight, page 379-387.

Virtual Tables: Creating a View

As you learned earlier, the output of a relational statement such as SELECT is another relation (or table). Suppose that at the end of each day, you would like to have a list of all products to reorder—that is, products with a quantity on hand that is less than or equal to the minimum quantity. Instead of typing the same query at the end of each day, wouldn't it be better to permanently save that query in the database? That is the function of a relational view. A view is a virtual table based on a SELECT query. The query can contain columns, computed columns, aliases, and aggregate functions from one or more tables. The tables on which the view is based are called **base tables**.

You can create a view by using the **CREATE VIEW** command:

CREATE VIEW *view-name* AS SELECT *query*

The CREATE VIEW statement is a data definition command that stores the subquery specification—the SELECT statement used to generate the virtual table—in the data dictionary.

A relational view has several special characteristics:

- You can use the name of a view anywhere a table name is expected in a SQL statement.
- Views are dynamically updated. That is, the view is recreated on demand each time it is invoked.
- Views provide a level of security in the database because they can restrict users to seeing only specified columns and rows in a table.
- Views may also be used as the basis for reports.

Read

Database Systems: Design, Implementation, and Management, 13th Edition Carlos Coronel and Steven Morris, Chapter Eight, page 387-388.

Chapter Summary/Review

In this chapter, you have learned:

- SQL commands can be divided into two overall categories: data definition language (DDL) commands and data manipulation language (DML) commands.
- The basic ANSI defined data types are NUMBER, NUMERIC, INTEGER, CHAR, VARCHAR, and DATE.
- The SELECT statement is the main data retrieval command in SQL. A SELECT statement has the following syntax:
  SELECT columnlist
  FROM tablelist
  [WHERE conditionlist]
  [GROUP BY columnlist]
  [HAVING conditionlist]
  [ORDER BY columnlist [ASC | DESC]];
- Operations that join tables can be classified as inner joins and outer joins. An inner join is the traditional join in which only rows that meet a given criterion are selected. An outer join returns the matching rows as well as the rows with unmatched attribute values for one table or both tables to be joined.
- A natural join returns all rows with matching values in the matching columns and eliminates duplicate columns. This style of query is used when the tables share a common attribute with a common name.
- Joins may use keywords such as USING and ON. If the USING clause is used, the query will return only the rows with matching values in the column indicated in the USING clause; that column must exist in both tables. If the ON clause is used, the query will return only the rows that meet the specified join condition.
- The ORDER BY clause is used to sort the output of a SELECT statement.
- The WHERE clause can be used with the SELECT, UPDATE, and DELETE statements to restrict the rows affected by the DDL command.
- Aggregate functions (COUNT, MIN, MAX, and AVG) are special functions that perform arithmetic computations over a set of rows, often used in conjunction with the GROUP BY clause to group the output of aggregate computations by one or more attributes.
- Subqueries and correlated queries are used when it is necessary to process data based on other processed data. That is, the query uses results that were previously unknown and that are generated by another query.

- Most subqueries are executed in a serial fashion. That is, the outer query initiates the data request, and then the inner subquery is executed. In contrast, a correlated subquery is a subquery that is executed once for each row in the outer query.

- SQL functions are used to extract or transform data. The most frequently used functions are date and time functions. The results of the function output can be used to store values in a database table, to serve as the basis for the computation of derived variables, or to serve as a basis for data comparisons.

- SQL provides relational set operators to combine the output of two queries to generate a new relation. The UNION and UNION ALL set operators combine the output of two or more queries. The INTERSECT relational set operator selects only the common rows. The EXCEPT (MINUS) set operator selects only the rows that are different.

- In order to successfully craft complex queries, the SQL programmer must understand the data with which he or she is working, and understand the problem to be solved. When struggling with the formulation of the query itself, building the query components in the order FROM, WHERE, GROUP BY, HAVING, SELECT, and ORDER BY can be helpful.

- The basic data definition commands allow you to create tables and indexes. The commands are CREATE TABLE, CREATE INDEX, ALTER TABLE, DROP TABLE, and DROP INDEX.

- The basic DML commands are SELECT, INSERT, UPDATE, and DELETE which allow you to retrieve, add, modify and delete rows from a table.

- The COMMIT and ROLLBACK commands are used to permanently save or reverse changes made to the rows. Once you COMMIT the changes, you cannot undo them with a ROLLBACK command.

- Views can be created to expose subsets of data to end users primarily for security and privacy reasons. Normally, views only store the SELECT statement to produce the view.

Review Questions

1. Explain why the following command would create an error and what changes could be made to fix the error:
   SELECT V_CODE, SUM(P_QOH) FROM PRODUCT;

2. What three join types are included in the outer join classification?

3. Explain the difference between an ORDER BY clause and a GROUP BY clause.

4. What is a subquery, and what are its basic characteristics?

5. What type of integrity is enforced when a primary key is declared?

6. Explain when an ALTER TABLE command might be needed.

7. What is the purpose of a CHECK constraint?

8. What is a view? Write its syntax.

Problems

9. Suppose a PRODUCT table contains two attributes, PROD_CODE and VEND_CODE. Those two attributes have values of ABC, 125, DEF, 124, GHI, 124, and JKL, 123, respectively. The VENDOR table contains a single attribute, VEND_CODE, with values 123, 124, 125, and 126, respectively. (The VEND_CODE attribute in the PRODUCT table is a foreign key to the VEND_CODE in the VENDOR table.) Given that information, what would be the query output for:

   a. A UNION query based on the two tables?
   b. A UNION ALL query based on the two tables?
   c. An INTERSECT query based on the two tables?
   d. An EXCEPT (MINUS) query based on the two tables?

The structure and contents of the Ch07_SaleCo database are shown in Figure P7.9. Use this database to answer the following problems.

FIGURE P7.9 THE CH07_SALECO DATABASE

Relational diagram

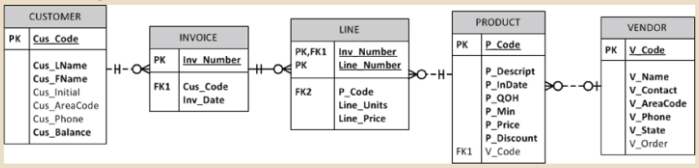


Table name: CUSTOMER

CUS_CODE	CUS_LNAME	CUS_FNAME	CUS_INITIAL	CUS_AREACODE	CUS_PHONE	CUS_BALANCE
10010	Ramas	Alfred	A	615	844-2573	0.00
10011	Dunne	Leona	K	713	894-1238	0.00
10012	Smith	Kathy	W	615	894-2285	345.86
10013	Olowski	Paul	F	615	894-2180	536.75
10014	Orlando	Myron		615	222-1672	0.00
10015	O'Brian	Amy	B	713	442-3381	0.00
10016	Brown	James	G	615	297-1228	221.19
10017	Williams	George		615	290-2556	768.93
10018	Farriss	Anne	G	713	382-7185	216.55
10019	Smith	Olette	K	615	297-3809	0.00

Table name: VENDOR

V_CODE	V_NAME	V_CONTACT	V_AREACODE	V_PHONE	V_STATE	V_ORDER
21225	Bryson, Inc.	Smithson	615	223-3234	TN	Y
21226	SuperLoo, Inc.	Flushing	904	215-8995	FL	N
21231	D&E Supply	Singh	615	228-3245	TN	Y
21344	Gomez Bros.	Ortega	615	889-2546	KY	N
22567	Dome Supply	Smith	901	678-1419	GA	N
23119	Randsets Ltd.	Anderson	901	678-3998	GA	Y
24004	Brackman Bros.	Browning	615	228-1410	TN	N
24288	ORDVA, Inc.	Hakford	615	898-1234	TN	Y
25443	B&K, Inc.	Smith	904	227-0093	FL	N
25501	Damal Supplies	Smythe	615	890-3529	TN	N
25595	Rubicon Systems	Orton	904	456-0092	FL	Y

Table name: INVOICE

INV_NUMBER	CUS_CODE	INV_DATE
1001	10014	16-Jan-18
1002	10011	16-Jan-18
1003	10012	16-Jan-18
1004	10011	17-Jan-18
1005	10018	17-Jan-18
1006	10014	17-Jan-18
1007	10015	17-Jan-18
1008	10011	17-Jan-18

Table name: LINE

INV_NUMBER	LINE_NUMBER	P_CODE	LINE_UNITS	LINE_PRICE
1001	1	13-Q2/P2	1	14.99
1001	2	23109-HB	1	9.95
1002	1	54778-2T	2	4.99
1003	1	2238/QPD	1	38.95
1003	2	1546-QQ2	1	39.95
1003	3	13-Q2/P2	5	14.99
1004	1	54778-2T	3	4.99
1004	2	23109-HB	2	9.95
1005	1	PVC23DRT	12	5.87
1006	1	SM-18277	3	6.99
1006	2	2232/QTY	1	109.92
1006	3	23109-HB	1	9.95
1006	4	89-WRE-Q	1	256.99
1007	1	13-Q2/P2	2	14.99
1007	2	54778-2T	1	4.99
1008	1	PVC23DRT	5	5.87
1008	2	WR3/TT3	3	119.95
1008	3	23109-HB	1	9.95

Table name: PRODUCT

P_CODE	P_DESCRIPT	P_INDATE	P_QOH	P_MIN	P_PRICE	P_DISCOUNT	V_CODE
11QER/31	Power painter, 15 psi., 3-nozzle	03-Nov-17	8	5	109.99	0.00	25595
13-Q2/P2	7.25-in. pwr. saw blade	13-Dec-17	32	15	14.99	0.05	21344
14-Q1/L3	9.00-in. pwr. saw blade	13-Nov-17	18	12	17.49	0.00	21344
1546-QQ2	Hrd. cloth, 1/4-in., 2x50	15-Jan-18	15	8	39.95	0.00	23119
1558-QWI	Hrd. cloth, 1/2-in., 3x50	15-Jan-18	23	5	43.99	0.00	23119
2232/QTY	B&D jigsaw, 12-in. blade	30-Dec-17	8	5	109.92	0.05	24288
2232/QWE	B&D jigsaw, 8-in. blade	24-Dec-17	6	5	99.87	0.05	24288
2238/QPD	B&D cordless drill, 1/2-in.	20-Jan-18	12	5	38.95	0.05	25595
23109-HB	Claw hammer	20-Jan-18	23	10	9.95	0.10	21225
23114-AA	Sledge hammer, 12 lb	02-Jan-18	8	5	14.40	0.05	
54778-2T	Rat-tail file, 1/8-in. fine	15-Dec-17	43	20	4.99	0.00	21344
89-WRE-Q	Hicut chain saw, 16 in.	07-Feb-18	11	5	256.99	0.05	24288
PVC23DRT	PVC pipe, 3.5-in., 8-ft	20-Feb-18	188	75	5.87	0.00	
SM-18277	1.25-in. metal screw, 25	01-Mar-18	172	75	6.99	0.00	21225
SW-23116	2.5-in. wd. screw, 50	24-Feb-18	237	100	8.45	0.00	21231
WR3/TT3	Steel matting, 4'x8'x1/6", .5" mesh	17-Jan-18	18	5	119.95	0.10	25595

10. Write a query to count the number of invoices.

11. Write a query to count the number of customers with a balance of more than $500.

12. Generate a listing of all purchases made by the customers. Sort the results by customer code, invoice number, and product description.

13. Using the output shown in Figure P7.12 as your guide, generate a list of customer purchases, including the subtotals for each of the invoice line numbers. The subtotal is a derived attribute calculated by multiplying LINE_UNITS by LINE_PRICE. Sort the output by customer code, invoice number, and product description. Be certain to use the column aliases as shown in the figure.

FIGURE P7.12 SUMMARY OF CUSTOMER PURCHASES WITH SUBTOTALS

CUS_CODE	INV_NUMBER	P_DESCRIPT	Units Bought	Unit Price	Subtotal
10011	1002	Rat-tail file, 1/8-in. fine	2	4.99	9.98
10011	1004	Claw hammer	2	9.95	19.90
10011	1004	Rat-tail file, 1/8-in. fine	3	4.99	14.97
10011	1008	Claw hammer	1	9.95	9.95
10011	1008	PVC pipe, 3.5-in., 8-ft	5	5.87	29.35
10011	1008	Steel matting, 4'x8'x1/6", .5" mesh	3	119.95	359.85
10012	1003	7.25-in. pwr. saw blade	5	14.99	74.95
10012	1003	B&D cordless drill, 1/2-in.	1	38.95	38.95
10012	1003	Hrd. cloth, 1/4-in., 2x50	1	39.95	39.95
10014	1001	7.25-in. pwr. saw blade	1	14.99	14.99
10014	1001	Claw hammer	1	9.95	9.95
10014	1006	1.25-in. metal screw, 25	3	6.99	20.97
10014	1006	B&D jigsaw, 12-in. blade	1	109.92	109.92
10014	1006	Claw hammer	1	9.95	9.95
10014	1006	Hicut chain saw, 16 in.	1	256.99	256.99
10015	1007	7.25-in. pwr. saw blade	2	14.99	29.98
10015	1007	Rat-tail file, 1/8-in. fine	1	4.99	4.99
10018	1005	PVC pipe, 3.5-in., 8-ft	12	5.87	70.44

Review Questions (MCQ)

1. The Count function in SQL returns the number of
    a. Values
    b. Distinct values
    c. Groups
    d. Columns

2. DML is provided for
    a. Description of logical structure of a database
    b. Addition of new structures in the database system
    c. Manipulation and processing of the database
    d. Definition of the physical structure of the database system

3. 'AS' clause is used in SQL for a
    a. Selection operation
    b. Rename operation
    c. Join operation
    d. Projection operation

4. The statement in SQL which allows to change the definition of a table is
    a. Alter
    b. Update
    c. Create
    d. Select

5. Which of the following is correct?
    a. A SQL query automatically eliminates duplicates
    b. SQL permits the same attribute names to be repeated in the same relation
    c. An SQL query will not work if there are no indexes on the relations
    d. None of these

6. Which of the following is a legal expression in SQL?
    a. SELECT NULL FROM EMPLOYEE;
    b. SELECT NAME FROM EMPLOYEE;
    c. SELECT NAME FROM EMPLOYEE WHERE SALARY = NULL;
    d. All of the above

7. The result of the UNION of T1 and T2 is a relation that includes
    a. All the tuples of T1
    b. All the tuples of T2
    c. All the tuples of T1 and T2
    d. All the tuples of T1 and T2 which have common columns

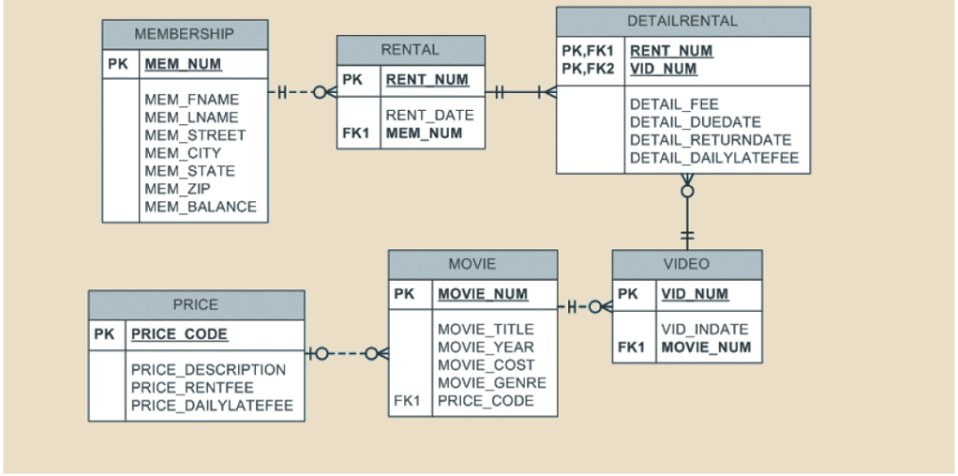
8. Which of the following is a comparison operator in SQL
    a. =
    b. LIKE
    c. BETWEEN
    d. All of the above

Case Studies / Projects

Case Study 4.1

EliteVideo is a start-up company providing concierge DVD kiosk service in upscale neighbourhoods. EliteVideo can own several copies (VIDEO) of each movie (MOVIE). For example, a kiosk may have 10 copies of the movie Twist in the Wind. In the database, Twist in the Wind would be one MOVIE, and each copy would be a VIDEO. A rental transaction (RENTAL) involves one or more videos being rented to a member (MEM-BERSHIP). A video can be rented many times over its lifetime; therefore, there is an M:N relationship between RENTAL and VIDEO. DETAILRENTAL is the bridge table to resolve this relationship. The complete ERD is provided in Figure P8.47.

FIGURE P8.47 THE CH08_MOVIECO ERD



Answer the following questions:

1. Write the SQL code to create the table structures for the entities shown in Figure P8.47. The structures should contain the attributes specified in the ERD. Use data types that are appropriate for the data that will need to be stored in each attribute. Enforce primary key and foreign key constraints as indicated by the ERD.
2. The following table provides a very small portion of the data that will be kept in the database. The data needs to be inserted into the appropriate table for testing purposes. Write the INSERT commands necessary to place the following data in the tables that were created in Problem 47. (If required by your DBMS, be certain to save the rows permanently.)

TABLE P8.48A

MEMBERSHIP TABLE

MEMBERSHIP							
MEM_NUM	MEM_FNAME	MEM_LNAME	MEM_STREET	MEM_CITY	MEM_STATE	MEM_ZIP	MEM_BALANCE
102	Tami	Dawson	2632 Takli Circle	Norene	TN	37136	11
103	Curt	Knight	4025 Cornell Court	Flatgap	KY	41219	6
104	Jamal	Melendez	788 East 145th Avenue	Quebeck	TN	38579	0
105	Iva	Mcclain	6045 Musket Ball Circle	Summit	KY	42783	15
106	Miranda	Parks	4469 Maxwell Place	Germantown	TN	38183	0
107	Rosario	Elliott	7578 Danner Avenue	Columbia	TN	38402	5

LEARNING OUTCOMES

After reading this Section of the guide, the learner should be able to:

- Give a detailed description of the role of database design as the foundation of a successful information system
- Describe the five phases in the Systems Development Life Cycle (SDLC)
- Design databases using the six phases in the Database Life Cycle (DBLC) framework
- Conduct evaluation and revision within the SDLC and DBLC framework
- Distinguish between top-down and bottom-up approached in database design
- Distinguish between centralized and decentralized conceptual database design

5.1 The Information System

Basically, a database is a carefully designed and constructed repository of facts. The data- base is part of a larger whole known as an information system (IS), which provides for data collection, storage, transformation, and retrieval. The information system also helps transform data into information, and it allows for the management of both data and information. Thus, a complete information system is composed of people, hardware, software, the database(s), application programs, and procedures. Systems analysis is the process that establishes the need for an information system and its extent. The process of creating an information system is known as systems development.

One key characteristic of current information systems is the strategic value of information in the age of global business. Therefore, information systems should always be aligned with strategic business mission and goals; the view of isolated and independent information systems is no longer valid. Current information systems should always be integrated with the company's enterprise-wide information systems architecture.

Read

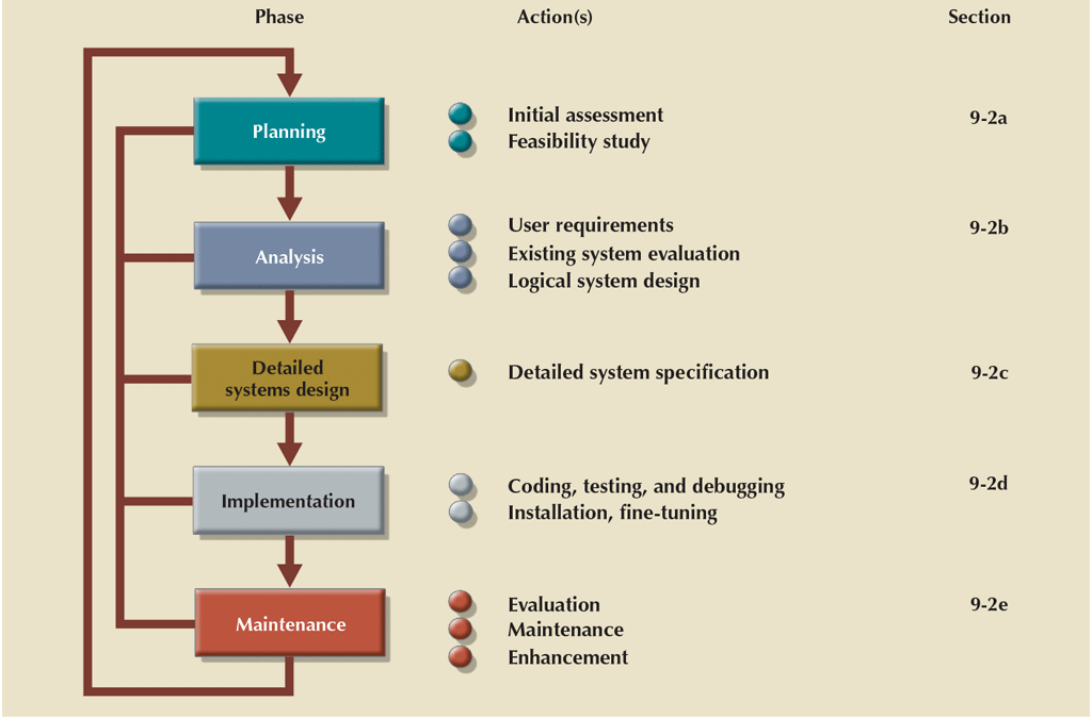
Database Systems: Design, Implementation, and Management, 13th Edition Carlos Coronel and Steven Morris, Chapter Nine, page 440-442.

5.2 The System Development Life Cycle (SDLC)

The Systems Development Life Cycle (SDLC) traces the history of an information system. Perhaps more important to the system designer, the SDLC provides the big picture within which the database design and application development can be mapped out and evaluated.

As illustrated in Figure 9.2, the traditional SDLC is divided into five phases: planning, analysis, detailed systems design, implementation, and maintenance. The SDLC is an iterative process rather than a sequential process. For example, the details of the feasibility study might help refine the initial assessment, and the details discovered during the user requirements portion of the SDLC might help refine the feasibility study.

FIGURE 9.2 THE SYSTEMS DEVELOPMENT LIFE CYCLE (SDLC)



Planning

The SDLC planning phase yields a general overview of the company and its objectives. An initial assessment of the information flow-and-extent requirements must be made during this discovery portion of the SDLC.

Even if you choose to "buy" rather than to "build," the system implementation must be carefully planned for it to be successful. Whatever the chosen option (build or buy), an analysis must be done to deploy the solution across the organization in ways that minimize cost and culture changes, while maximizing value. The SDLC provides a framework for sound planning and implementation.

Analysis

Problems defined during the planning phase are examined in greater detail during the analysis phase. A macro analysis must be made both of individual needs and organizational needs, addressing questions such as:

- What are the requirements of the current system's end users?
- Do those requirements fit into the overall information requirements? The analysis phase of the SDLC is, in effect, a thorough audit of user requirements. The existing hardware and software systems are also studied during the analysis phase.

The result of the analysis should be a better understanding of the system's functional areas, actual and potential problems, and opportunities.

End users and the system designer(s) must work together to identify processes and uncover potential problem areas. Such cooperation is vital to defining the appropriate performance objectives by which the new system can be judged.

Along with a study of user requirements and the existing systems, the analysis phase also includes the creation of a logical systems design. The logical design must specify the appropriate conceptual data model, inputs, processes, and expected output requirements. When creating a logical design, the designer might use tools such as data flow diagrams (DFDs), hierarchical input process output (HIPO) diagrams, entity relation- ship (ER) diagrams, and even some application prototypes. The database design's data-modelling activities take place at this point to discover and describe all entities and their attributes, and the relationships among the entities within the database.

Defining the logical system also yields functional descriptions of the system's components (modules) for each process within the database environment. All data transformations (processes) are described and documented, using systems analysis tools such as DFDs. The conceptual data model is validated against those processes.

Detailed System Design

In the detailed systems design phase, the designer completes the design of the system's processes. The design includes all the necessary technical specifications for the screens, menus, reports, and other devices that might help make the system a more efficient information generator. The steps are laid out for conversion from the old system to the new system. Training principles and methodologies are also planned and must be submitted for management's approval.

Implementation

During the implementation phase, the hardware, DBMS software, and application programs are installed, and the database design is implemented. During the initial stages of the implementation phase, the system enters a cycle of coding, testing, and debugging until it is ready to be delivered. The actual database is created, and the system is customized by the creation of tables and views, user authorizations, and so on.

The database contents might be loaded interactively or in batch mode, using a variety of methods and devices:

- Customized user programs
- Database interface programs
- Conversion programs that import the data from a different file structure, using batch
- Programs, a database utility, or both the system is subjected to exhaustive testing until it is ready for use. Traditionally, the implementation and testing of a new system took 50 to 60 percent

Maintenance

Almost as soon as the system is operational, end users begin to request changes in it. Those changes generate system maintenance activities, which can be grouped into three types:

- Corrective maintenance in response to systems errors
- Adaptive maintenance due to changes in the business environment
- Perfective maintenance to enhance the system

Because every request for structural change requires retracing the SDLC steps, the system is, in a sense, always at some stage of the SDLC.

Each system has a predetermined operational life span, but its actual life span depends on its perceived utility. There are several reasons for reducing the operational life of certain systems. Rapid technological change is one reason, especially for systems based on processing speed and expandability. Another common reason is the cost of maintaining a system.

If the system's maintenance cost is high, its value becomes suspect. Computer-aided software engineering (CASE) tools, such as System Architect or Visio Professional, help produce better systems within a reasonable amount of time and at a reasonable cost. In addition, CASE-produced applications are more structured, better documented, and especially standardized, which tends to prolong the operational life of systems by making them easier and cheaper to update and maintain

Read

Database Systems: Design, Implementation, and Management, 13th Edition Carlos Coronel and Steven Morris, Chapter Nine, page 442-445.

5.3 The Database Life Cycle

Within the larger information system, the database is subject to a life cycle as well. The Database Life Cycle (DBLC) contains six phases, as shown in Figure 9.3: database initial study, database design, implementation and loading, testing and evaluation, operation, and maintenance and evolution.

The Database Initial Study

If a designer has been called in, chances are that the current system has failed to perform functions deemed vital by the company. (You don't call the plumber unless the pipes leak.) Therefore, in addition to examining the current system's operation within the company, the designer must determine how and why the current system fails. That means spending a lot of time talking and listening to end users. Although database design is a technical business, it is also people-oriented. Database designers must be excellent communicators and must have finely tuned interpersonal skills.

Depending on the complexity and scope of the database environment, the database designer might be a lone operator or part of a systems development team composed of a project leader, one or more senior systems analysts, and one or more junior systems analysts. The word designer is used generically here to cover a wide range of design team compositions. Computer aided systems engineering (CASE) Tools used to automate part or all the Systems Development Life Cycle. Database Life Cycle (DBLC) A cycle that traces the history of a database within an information system. The cycle is divided into six phases: initial study, design, implementation and loading, testing and evaluation, operation and maintenance, and evolution.

The overall purpose of the database initial study is to:

- Analyse the company situation
- Define problems and constraints
- Define objectives
- Define scope and boundaries

Figure 9.4 depicts the interactive and iterative processes required to complete the first phase of the DBLC successfully. Note that the database initial study phase leads to the development of database system objectives. Using Figure 9.4 as a discussion template, examine each of its components in greater detail.

Analyse the Company Situation the company situation describes the general conditions in which a company operates, its organizational structure, and its mission. To analyse the company situation, the database designer must learn the company's operational components, how they function, and how they interact.

The following issues must be resolved:

- What is the organization's general operating environment, and what is its mission within that environment?
- What is the organization's structure? Knowing who controls what and who reports to whom is quite useful when you need to define required information flows, specific report and query formats, and so on.

Define Problems and Constraints the designer has both formal and informal sources of information. If the company has existed for any length of time, it already has a system in place (either manual or computer-based). How does the existing system function? What input does the system require? What documents does the system generate? By whom and how is the system output used? Studying the paper trail can be very informative. In addition to the official version of the system's operation, there is also the more informal, perhaps more real version; the designer must be shrewd enough to see how these differ.

The process of defining problems might initially appear to be unstructured. Company end users often cannot precisely describe the larger scope of company operations or identify the real problems encountered during company operations. Often the managerial view of a company's operation and its problems is different from that of the end users, who perform the actual routine work

During the initial problem definition process, the designer is likely to collect very broad problem descriptions. For example, note the following concerns expressed by the president of a fast-growing, transnational manufacturing company:

Although the rapid growth is gratifying, members of the management team are concerned that such growth is beginning to undermine the ability to maintain a high customer service standard, and perhaps worse, to diminish manufacturing standards control.

The problem definition process quickly leads to a host of general problem descriptions. For example, the marketing manager comments:

I'm working with an antiquated filing system. We manufacture more than 1,700 specialty machine parts. When a regular customer calls in, we can't get a very quick inventory scan. If a new customer calls in, we can't do a current parts search by using a simple description, so we often do a machine setup for a part that we have in inventory. That's wasteful. And of course, some new customers get irritated when we can't give a quick response.

Finding precise answers is important, especially concerning the operational relation- ships among business units. If a proposed system will solve the marketing department's problems but exacerbate

those of the production department, not much progress will have been made. Using an analogy, suppose that your home water bill is too high. You have determined the problem: the faucets leak. The solution? You step outside and cut off the water supply to the house. However, is that an adequate solution, or would the replacement of faucet washers do a better job of solving the problem? You might find this scenario simplistic, yet almost any experienced database designer can find similar instances of database problem solving, although they are admittedly more complicated.

Database Design

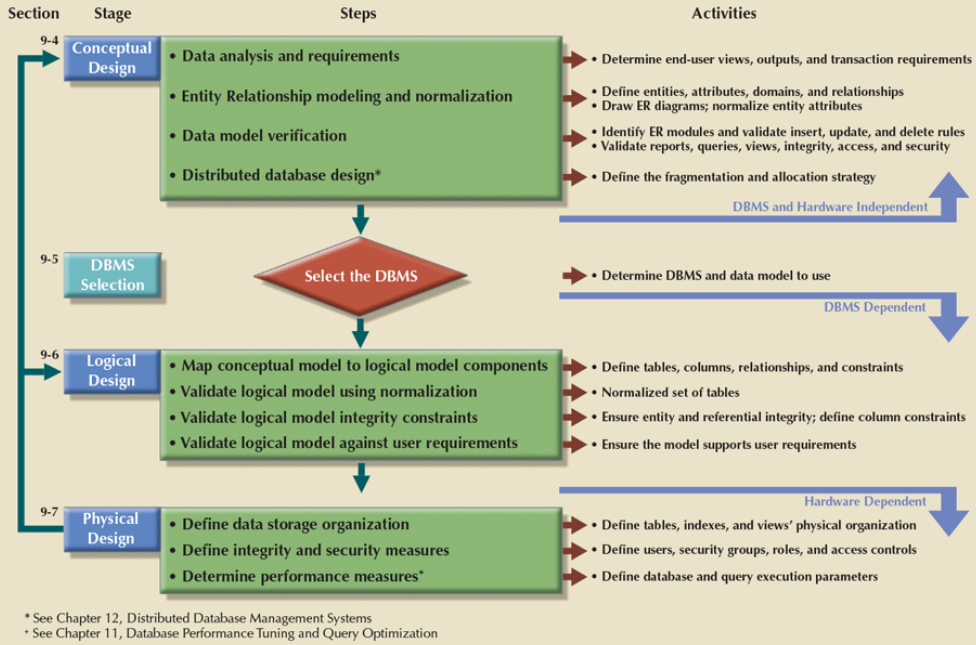
The second phase of the DBLC focuses on the design of the database model that will support company operations and objectives. This is arguably the most critical DBLC phase: making sure that the final product meets user and system requirements. In the process of database design, you must concentrate on the data characteristics required to build the database model. At this point, there are two views of the data within the system: the business view of data as a source of information and the designer's view of the data structure, its access, and the activities required to transform the data into information. Figure 9.5 contrasts those views. Note that you can summarize the different views by looking at the terms what and how. Defining data is an integral part of the DBLC's second phase.

As you examine the procedures required to complete the design phase in the DBLC, remember these points:

- The process of database design is loosely related to the analysis and design of a larger system. The data component is only one element of a larger information system.
- The systems analysts or systems programmers are in charge of designing the other system components. Their activities create the procedures that will help transform the data within the database into useful information.
- The database design does not, constitute a sequential process. Rather, it is an iterative process that provides continuous feedback designed to trace previous steps.

The database design process is depicted in Figure 9.6. The figure shows that there are three essential stages: conceptual, logical, and physical design, plus the DBMS selection decision, which is critical to determine the type of logical and physical designs to be created. The design process starts with conceptual design and moves to the logical and physical design stages. At each stage, more details about the data model design are determined and documented. You could think of the conceptual design as the overall data as seen by the end user, the logical design as the data as seen by the DBMS, and the physical design as the data as seen by the operating system's storage management devices.

FIGURE 9.6 DATABASE DESIGN PROCESS



It is important to note that the overwhelming majority of database designs and implementations are based on the relational model, and therefore use the relational model constructs and techniques. When you finish the design activities, you will have a complete database design ready to be implemented.

Implementation and Loading

The output of the database design phase is a series of instructions detailing the cre- ation of tables, attributes, domains, views, indexes, security constraints, and storage and performance guidelines. In this phase, you actually implement all these design specifications.

Install the DBMS This step is required only when a new dedicated instance of the DBMS is necessary for the system. In many cases, the organization will have made a particular DBMS the standard to leverage investments in the technology and the skills that employees have already developed. The DBMS may be installed on a new server or on existing servers. One current trend is called virtualization. Virtualization is a technique that creates logical representations of computing resources that are independent of the underlying physical computing resources. This technique is used in many areas of computing, such as the creation of virtual servers, virtual storage, and virtual private networks. In a database environment, database virtualization refers to the installation of a new instance of the DBMS on a virtual server running on shared hardware. This is normally a task that involves system and network administrators to create appropriate user groups and services in the server configuration and network routing. Another common trend is the use of cloud database services such Microsoft SQL Database Service or Amazon Relational Database Services (RDS). This new generation of services allows user to create databases that could be easily managed, tested, and scaled up as needed.

Create the Database(s) in most modern relational DBMSs, a new database implementation requires the creation of special storage-related constructs to house the end- user tables. The constructs usually include the storage group (or file groups), the table spaces, and the tables.

Load or Convert the Data After the database has been created, the data must be loaded into the database tables. Typically, the data will have to be migrated from the prior version of the system. Often, data to be included in the system must be aggregated from multiple sources. In a best-case scenario, all the data will be in a relational database so that it can be readily transferred to the new database. However, in some cases data may have to be imported from other relational databases, no relational databases, flat files, legacy systems, or even manual paper-and-pencil systems. If the data format does not support direct importing into the new database, conversion programs may have to be created to reformat the data for importing. In a worst-case scenario, much of the data may have to be manually entered into the database. Once the data has been loaded, the DBA works with the application developers to test and evaluate the database.

Loading existing data into a cloud-based database service sometimes can be expensive. The reason for this is that most cloud services are priced based not only on the volume of data to be stored but also on the amount of data that travels over the network. In such cases, loading a 1 TB database could be a very expensive proposition. Therefore, system administrators must be very careful in reading and negotiating the terms of cloud service contracts to ensure that there will be no “hidden” costs.

Testing and Evaluating

In the design phase, decisions were made to ensure integrity, security, performance, and recoverability of the database. During implementation and loading, these plans were put into place In testing and evaluation, the DBA tests and fine-tunes the database to ensure that it performs as expected. This phase occurs in conjunction with application programming. Programmers use database tools to prototype the applications during coding of the programs. Tools such as report generators, screen painters, and menu generators are especially useful to application programmers.

Test the Database During this step, the DBA tests the database to ensure that it maintains the integrity and security of the data. Data integrity is enforced by the DBMS through the proper use of primary and foreign key rules. Many DBMSs also support the creation of domain constraints and database triggers. Testing will ensure that these constraints were properly designed and implemented. Data integrity is also the result of properly implemented data management policies, which are part of a comprehensive data administration framework.

Previously, users and roles were created to grant users access to the data. In this stage, not only must those privileges be tested, but the broader view of data privacy and security must be addressed. Data stored in the company database must be protected from access by unauthorized users. (It does not take much imagination to predict the likely results if students have access to a student database or if employees have access to payroll data!) Consequently, you must test for at least the following:

- Physical security allows only authorized personnel physical access to specific areas. Depending on the type of database implementation, however, establishing physical security might not always be practical. For example, a university student research database is not a likely candidate for physical security.
- Password security allows the assignment of access rights to specific authorized users. Password security is usually enforced at login time at the operating system level.
- Access rights can be established through the database software. The assignment of access rights may restrict operations (CREATE, UPDATE, DELETE, and so on) on predetermined objects such as databases, tables, views, queries, and reports.

- Audit trails are usually provided by the DBMS to check for access violations. Although the audit trail is an after-the-fact device, its mere existence can discourage unauthorized use.
- Data encryption can render data useless to unauthorized users who might have violated some of the database security layers.
- Diskless workstations allow end users to access the database without being able to download the information from their workstations.

Fine-tune the Database Database performance can be difficult to evaluate because there are no standards for measuring it, but it is typically one of the most important factors in database implementation. Different systems will place different requirements on the database. Systems that support rapid transactions will require the database to be implemented so that they provide superior performance during high volumes of inserts, updates, and deletes. Other systems, like decision support systems, may require superior performance for complex data retrieval tasks. Many factors can affect the database's performance on various tasks, including the hardware and software environment in which the database exists. Naturally, the characteristics and volume of the data also affect database performance: a search of 10 tuples is faster than a search of 100,000 tuples. Other important factors in database performance include system and database configuration parameters such as data placement, access path definition, the use of indexes, and buffer size. For a more in-depth discussion of database performance issues, see Chapter 11, Database Performance Tuning and Query Optimization.

Evaluate the Database and Its Application Programs

As the database and application programs are created and tested, the system must also be evaluated using a more holistic approach. Testing and evaluation of the individual components should culminate in a variety of broader system tests to ensure that all of the components interact properly to meet the needs of the users. At this stage, integration issues and deployment plans are refined, user training is conducted, and system documentation is finalized. Once the system receives final approval, it must be a sustainable resource for the organization. To ensure that the data contained in the database is protected against loss, backup and recovery plans are tested.

Timely data availability is crucial for almost every database. Unfortunately, the data- base can lose data through unintended deletions, power outages, and other causes. Data backup and recovery procedures create a safety valve, ensuring the availability of consistent data. Typically, database vendors encourage the use of fault-tolerant components such as uninterruptible power supply (UPS) units, RAID storage devices, clustered servers, and data replication technologies to ensure the continuous operation of the database in case of a hardware failure. Even with these components, backup and restore functions constitute a very important part of daily database operations. Some DBMSs provide functions that allow the database administrator to schedule automatic database backups to permanent storage devices such as disks, DVDs, tapes, and online storage. Database backups can be performed at different levels:

- A full backup, or dump, of the entire database. In this case, all database objects are backed up in their entirety.
- A differential backup of the database, in which only the objects that have been updated or modified since the last full backup are backed up.

- A transaction log backup, which backs up only the transaction log operations that are not reflected in a previous backup copy of the database. In this case, no other database objects are backed up. (For a complete explanation of the transaction log, see Chapter 10, Transaction Management and Concurrency Control.)

The database backup is stored in a secure place, usually in a different building from the database itself, and is protected against dangers such as fire, theft, flood, and other potential calamities. The main purpose of the backup is to guarantee database restoration following a hardware or software failure.

Failures that plague databases and systems are generally induced by software, hard- ware, programming exemptions, transactions, or external factors. Table 9.1 summarizes the most common sources of database failure.

Depending on the type and extent of the failure, the recovery process ranges from a minor short-term inconvenience to a major long-term rebuild. Regardless of the extent of the required recovery process, recovery is not possible without a usable backup.

Database recovery generally follows a predictable scenario. First, the type and extent of the required recovery are determined. If the entire database needs to be recovered to a consistent state, the recovery uses the most recent backup copy of the database in a known consistent state. The backup copy is then rolled forward to restore all subsequent transactions by using the transaction log information. If the database needs to be recovered but the committed portion of the database is still usable, the recovery process uses the transaction log to "undo" all of the transactions that were not committed (see Chapter 10, Transaction Management and Concurrency Control).

At the end of this phase, the database completes an iterative process of testing, evaluation, and modification that continues until the system is certified as ready to enter the operational phase.

Operation

Once the database has passed the evaluation stage, it is considered operational. At that point, the database, its management, its users, and its application programs constitute a complete information system.

The beginning of the operational phase invariably starts the process of system evolution. As soon as all the targeted end users have entered the operations phase, problems that could not have been foreseen during the testing phase begin to surface. Some of the problems are serious enough to warrant emergency "patchwork," while others are merely minor annoyances.

The web, the sheer volume of transactions might cause even a well-designed system to bog down. In that case, the designers must identify the source of the bottleneck and produce alternative solutions. Those solutions may include using load-balancing soft- ware to distribute the transactions among multiple computers, increasing the available cache for the DBMS, and so on. The demand for change is the designer's constant concern, which leads to phase 6, maintenance and evolution.

Maintenance and Evolution

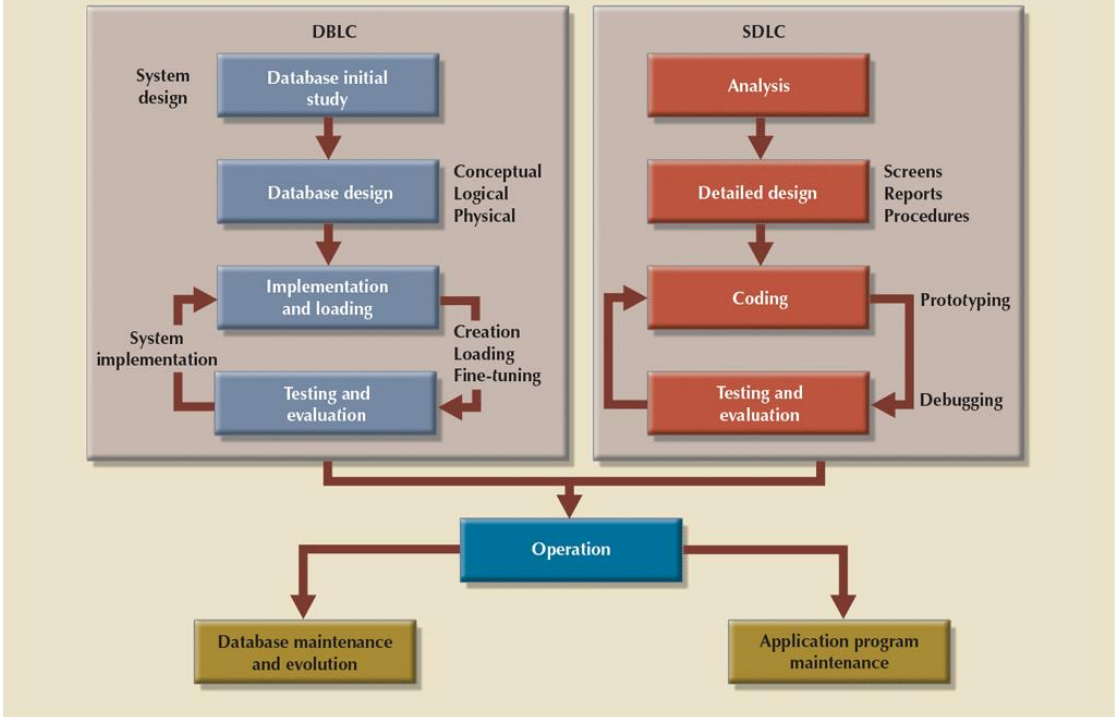
The database administrator must be prepared to perform routine maintenance activities within the database. Some of the required periodic maintenance activities include:

- Preventive maintenance (backup)
- Corrective maintenance (recovery)
- Adaptive maintenance (enhancing performance, adding entities and attributes, and so on)
- Assignment of access permissions and their maintenance for new and old users
- Generation of database access statistics to improve the efficiency and usefulness of system audits and to monitor system performance
- Periodic security audits based on the system-generated statistics
- Monthly, quarterly, or yearly system usage summaries for internal billing or budgeting purposes

The likelihood of new information requirements and the demand for additional reports and new query formats require application changes and possible minor changes in the database components and contents. These changes can be easily implemented only when the database design is flexible and when all documentation is updated and online. Eventually, even the best-designed database environment will no longer be capable of incorporating such evolutionary changes, and then the whole DBLC process begins anew.

As you can see, many of the activities described in the DBLC are similar to those in the SDLC. This should not be surprising because the SDLC is the framework within which the DBLC activities take place. A summary of the parallel activities that occur within the SDLC and DBLC is shown in Figure 9.8.

FIGURE 9.8 PARALLEL ACTIVITIES IN THE DBLC AND THE SDLC



Read

Database Systems: Design, Implementation, and Management, 13th Edition Carlos Coronel and Steven Morris, Chapter Nine, page 445-457.

5.4 Conceptual Design

Recall that the second phase of the DBLC is database design, which comprises three stages: conceptual design, logical design, and physical design, plus the critical decision of DBMS selection. Conceptual design is the first stage in the database design process. The goal at this stage is to design a database that is independent of database software and physical details. The output of this process is a conceptual data model that describes the main data entities, attributes, relationships, and constraints of a given problem domain. This design is descriptive and narrative in form. In other words, it is generally composed of a graphical representation as well as textual descriptions of the main data elements, relationships, and constraints. In this stage, data modelling is used to create an abstract database structure that rep-resents real-world objects in the most realistic way possible. The conceptual model must embody a clear understanding of the business and its functional areas. At this level of abstraction, the type of hardware and database model to be used might not have been identified yet. Therefore, the design must be software-and hardware-independent so that the system can be set up within any platform chosen later.

Keep in mind the following minimal data rule: All that is needed is there, and all that is there is needed. In other words, make sure that all data needed is in the model and that all data in the model is needed. All data elements required by the database transactions must be defined in the model, and all data elements defined in the model must be used by at least one database transaction.

However, as you apply the minimal data rule, avoid excessive short-term bias. Focus not only on the immediate data needs of the business but on future data needs. Thus, the database design must leave room for future modifications and additions, ensuring that the business's investment in information resources will endure. The conceptual design has four steps, which are listed in Table 9.2.

TABLE 9.2

CONCEPTUAL DESIGN STEPS

STEP	ACTIVITY
1	Data analysis and requirements
2	Entity relationship modeling and normalization
3	Data model verification
4	Distributed database design

Data Analysis and Requirements

The first step in conceptual design is to discover the characteristics of the data elements. An effective database is an information factory that produces key ingredients for successful decision making. Appropriate data element characteristics are those that can be transformed into appropriate information. Therefore, the designer's efforts are focused on:

- Information needs. What kind of information is needed?
- Information users. Who will use the information? How is the information to be used? What are the various end-user data views?
- Information sources. Where is the information to be found? How is the information to be extracted once it is found?
- Information constitution. What data elements are needed to produce the information? What are the data attributes? What relationships exist in the data? What is the data volume?
- The designer obtains the answers to those questions from a variety of sources to compile the necessary information:
- Developing and gathering end-user data views. The database designer and the end user(s) jointly develop a precise description of end-user data views, which in turn are used to help identify the database's main data elements.
- Directly observing the current system: existing and desired output. The end user usually has an existing system in place, whether it is manual or computer-based. The designer reviews the existing system to identify the data and its characteristics. The designer examines the input forms and files (tables) to discover the data type and volume. If the end user already has an automated system in place, the designer carefully examines the current and desired reports to describe the data required to support the reports.
- Interfacing with the systems design group. As noted earlier in this chapter, the database design process is part of the SDLC. In some cases, the systems analyst in charge of designing the new system will also develop the conceptual database model. (This is usually true in a decentralized environment.) In other cases, the database design is considered part of the DBA's job. The presence of a DBA usually implies the existence of a formal data-processing department.

The DBA designs the database according to the specifications created by the systems analyst.

To develop an accurate data model, the designer must have a thorough understanding of the company's data types and their extent and uses. But data does not, by itself, yield the required understanding of the total business. From a database point of view, the collection of data becomes meaningful only when business rules are defined. Remember from Chapter 2, Data Models, that a business rule is a brief and precise description of a policy, procedure, or principle within a specific organization's environment. Business rules, derived from a detailed description of an organization's operations, help to create and enforce actions within that organization's environment. When business rules are written properly, they define entities, attributes, relationships, connectivity's, cardinalities, and constraints.

To be effective, business rules must be easy to understand, and they must be widely disseminated to ensure that every person in the organization shares a common interpretation of the rules. Using simple language, business rules describe the main and distinguishing characteristics of the data as viewed by the company. Examples of business rules are as follows:

- A customer may make many payments on an account.
- Each payment on an account is credited to only one customer.
- A customer may generate many invoices.
- Each invoice is generated by only one customer.

Given their critical role in database design, business rules must not be established casually. Poorly defined or inaccurate business rules lead to database designs and implementations that fail to meet the needs of the organization's end users.

Ideally, business rules are derived from a formal description of operations, which is a document that provides a precise, up-to-date, and thoroughly reviewed description of the activities that define an organization's operating environment. (To the database designer, the operating environment is both the data sources and the data users.) Naturally, an organization's operating environment is dependent on the organization's mission. For example, the operating environment of a university would be quite different from that of a steel manufacturer, an airline, or a nursing home. Yet, no matter how different the organizations may be, the data analysis and requirements component of the database design is enhanced when the data environment and data use are described accurately and precisely within a description of operations.

In a business environment, the main sources of information for the description of operations—and therefore of business rules—are company managers, policymakers, department managers, and written documentation such as company procedures, standards, and operations manuals. A faster and more direct source of business rules is direct interviews with end users. Unfortunately, because perceptions differ, the end user can be 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 should perform such a task. This distinction might seem trivial, but it has major legal consequences. Although end users are crucial contributors to the development of business rules, it pays to verify end-user perceptions. Often, interviews with several people who perform the same job yield very different perceptions of their job components. While such a discovery might point to "management problems," that general diagnosis does not help the database designer. Given the discovery of such problems, the database designer's job is to reconcile the differences and verify the results of the reconciliation to ensure that the business rules are appropriate and accurate. Knowing the business rules enables the designer to fully understand how the description of operations A document that provides a precise, detailed, up-to-date, and thoroughly reviewed description of the activities that define an organization's operating environment. ness works and what role the data plays within company operations. Consequently, the designer must identify the company's business rules and analyse their impact on the nature, role, and scope of data. Business rules yield several important benefits in the design of new systems:

- They help standardize the company's view of data.
- They constitute 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 foreign key constraints. See Chapter 4, Entity Relationship (ER) Medellin

The last point is especially noteworthy: whether a given relationship is mandatory or optional is usually a function of the applicable business rule.

Entity Relationship Modelling and Normalization

Before creating the ER model, the designer must communicate and enforce appropriate standards to be used in the documentation of the design. The standards include the use of diagrams and symbols, documentation writing style, layout, and any other conventions to be followed during documentation. Designers often overlook this very import-ant requirement, especially when they are working as members of a design team. Failure to standardize documentation often means a failure to communicate later, and communications failures often lead to poor design work. In

contrast, well-defined and enforced standards make design work easier and promise (but do not guarantee) a smooth integration of all system components. Because the business rules usually define the nature of the relationship(s) among the entities, the designer must incorporate them into the conceptual model. The process of defining business rules and developing the conceptual model using ER diagrams can be described using the steps shown in Table 9.3.6

Developing the Conceptual model using some of the steps listed in Table 9.3 take place concurrently, and some, such as the normalization process, can generate a demand for additional entities and/or attributes, thereby causing the designer to revise the ER model.

As you will likely discover, the initial ER model may be subjected to several revisions before it meets the system's requirements. Such a revision process is quite natural. Remember that the ER model is a communications tool as well as a design blueprint. Therefore, when you meet with the proposed system users, the initial ER model should give rise to questions such as "Is this really what you meant?"

Data Model Verification

From the preceding discussion, you might get the impression that ER modelling activities such as entity and attribute definition, normalization, and verification take place in a precise sequence. In fact, once you have completed the initial ER model, chances are that you will move back and forth among the activities until you are satisfied that the ER model accurately represents a database design that can meet the required system demands. The activities often take place in parallel, and the process is iterative. All objects (entities, attributes, relations, views, and so on) are defined in a data dictionary, which is used in tandem with the normalization process to help eliminate data anomalies and redundancy problems. During this ER modelling process, the designer must:

- Define entities, attributes, primary keys, and foreign keys. (The foreign keys serve as the basis for the relationships among the entities.)
- Make decisions about adding new primary key attributes to satisfy end-user and processing requirements.
- Make decisions about the treatment of composite and multivalued attributes.
- Make decisions about adding derived attributes to satisfy processing requirements.
- Make decisions about the placement of foreign keys in 1:1 relationships
- Avoid unnecessary ternary relationships.
- Draw the corresponding ER diagram.
- Normalize the entities.
- Include all data element definitions in the data dictionary.
- Make decisions about standard naming conventions.

The naming conventions requirement is important, yet it is frequently ignored at the designer's risk. Real database design is generally accomplished by teams. Therefore, it is important to ensure that team members work in an environment in which naming standards are defined and enforced. Proper documentation is crucial to the successful completion of the design, and adherence to the naming conventions serves database designers well. In fact, a common refrain from users seems to be: "I didn't know why you made such a fuss over naming conventions, but now that I'm doing this stuff for real, I've become a true believer."

Data model verification is one of the last steps in the conceptual design stage, and it is one of the most critical. In this step, the ER model must be verified against the proposed system processes to corroborate that they can be supported by the database model. Verification requires that the model be run through a series of tests against:

- End-user data views All required transactions: SELECT, INSERT, UPDATE, and DELETE operations
- Access rights and security
- Business-imposed data requirements and constraints

Because real-world database design is generally done by teams, the database design is probably divided into major components known as modules. A module is an information system component that handles a specific business function, such as inventory, orders, or payroll. Under these conditions, each module is supported by an ER segment that is a subset or fragment of an enterprise ER model. Working with modules accomplishes several important ends:

- The modules (and even the segments within them) can be delegated to design groups within teams, greatly speeding up the development work.
- The modules simplify the design work. The large number of entities within a complex design can be daunting. Each module contains a more manageable number of entities. Module (1) a design segment that can be implemented as an autonomous unit, and is sometimes linked to produce a system. (2) An information system component that handles a specific function, such as inventory, orders, or payroll.
- The modules can be prototyped quickly. Implementation and application programming trouble spots can be identified more readily. Quick prototyping is also a great confidence builder.
- Even if the entire system cannot be brought online quickly, the implementation of one or more modules will demonstrate that progress is being made and that at least part of the system is ready to begin serving the end users.

As useful as modules are, they represent a loose collection of ER model fragments that could wreak havoc in the database if left unchecked. For example, the ER model fragments:

- Might present overlapping, duplicated, or conflicting views of the same data
- Might not be able to support all processes in the system's module

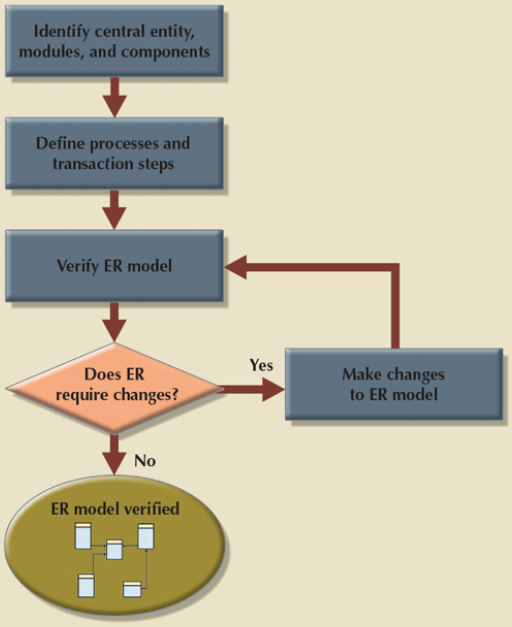
To avoid these problems, it is better if the modules' ER fragments are merged into a single enterprise ER model. This process starts by selecting a central ER model segment and iteratively adding more ER model segments one at a time. At each stage, for each new entity added to the model, you need to validate that the new entity does not overlap or conflict with a previously identified entity in the enterprise ER model. Merging the ER model segments into an enterprise ER model triggers a careful revaluation of the entities, followed by a detailed examination of the attributes that describe those entities. This process serves several important purposes:

- The emergence of the attribute details might lead to a revision of the entities themselves. Perhaps some of the components first believed to be entities will instead turn out to be attributes within other entities. Or, a component that was originally considered an attribute might turn out to contain a sufficient number of subcomponents to warrant the introduction of one or more new entities.
- The focus on attribute details can provide clues about the nature of relationships as they are defined by the primary and foreign keys. Improperly defined relationships lead to implementation problems first and to application development problems later.
- To satisfy processing and end-user requirements, it might be useful to create a new primary key to replace an existing primary key. For example, in the example illustrated in Figure 9.9, a surrogate primary key (RENTAL_ID) could be introduced to replace the original primary key composed of VIDEO_ID and CUST_NUM.
- Unless the entity details (the attributes and their characteristics) are precisely defined, it is difficult to evaluate the extent of the design's normalization. Knowledge of the normalization levels helps guard against undesirable redundancies.

- A careful review of the rough database design blueprint is likely to lead to revisions. Those revisions will help ensure that the design is capable of meeting end-user requirements.

Keep in mind that this process requires the continuous verification of business trans-actions as well as system and user requirements. The verification sequence must be repeated for each of the system's modules. Figure 9.12 illustrates the iterative nature of the process.

FIGURE 9.12 ITERATIVE ER MODEL VERIFICATION PROCESS



The verification process starts with selecting the central (most important) entity, which is the focus for most of the system's operations. To identify the central entity, the designer selects the entity involved in the greatest number of the model's relationships. In the ER diagram, it is the entity with more lines connected to it than any other. The next step is to identify the module or subsystem to which the central entity belongs and to define that module's boundaries and scope. The entity belongs to the module that uses it most frequently. Once each module is identified, the central entity is placed within the module's framework to let you focus on the module's details. Within the central entity/module framework, you must

- Ensure the module's cohesivity. The term cohesivity describes the strength of the relation-ships found among the module's entities. A module must display high cohesivity—that is, the entities must be strongly related, and the module must be complete and self-sufficient. Cohesively The strength of the relationships between a module's components. Module cohesivity must be high. Module coupling the extent to which modules are independent of one another.
- Analyse each module's relationships with other modules to address module coupling. module coupling describes the extent to which modules are independent of one another. Modules must display low coupling, indicating that they are independent of other modules. Low coupling decreases unnecessary intermodal dependencies, thereby allowing the creation of a truly modular system and eliminating unnecessary relationships among entities. Processes may be classified according to their:
- Frequency (daily, weekly, monthly, yearly, or exceptions)
- Operational type (INSERT or ADD, UPDATE or CHANGE, DELETE, queries and reports, batches, maintenance, and backups)

All identified processes must be verified against the ER model. If necessary, appropriate changes are implemented. The process verification is repeated for all of the model's modules. You can expect that additional entities and attributes will be incorporated into the conceptual model during its validation. At this point, a conceptual model has been defined as hardware-and software-independent. Such independence ensures the system's portability across platforms. Portability can extend the database's life by making it possible to migrate to another DBMS and hardware platform Distributed Database Design

Read

Database Systems: Design, Implementation, and Management, 13th Edition Carlos Coronel and Steven Morris, Chapter Nine, page 457-467.

DBMS Software Selection

The selection of DBMS software is critical to the information system's smooth operation. Consequently, the advantages and disadvantages of the proposed DBMS software should be carefully studied. To avoid false expectations, the end user must be made aware of the limitations of both the DBMS and the database. Although the factors that affect the purchasing decision vary from company to company, some of the most common are:

Although the factors that affect the purchasing decision vary from company to company, some of the most common are:

- *Cost*. This includes the original purchase price, along with maintenance, operational, license, installation, training, and conversion costs.
- *DBMS features and tools*. Some database software includes a variety of tools that facilitate application development.
- *Underlying model*. This can be hierarchical, network, relational, object/relational, or object-oriented.
- *Portability*. A DBMS can be portable across platforms, systems, and languages.
- *DBMS hardware requirements*. Items to consider include processor(s), RAM, disk space, and so on.

Read

Database Systems: Design, Implementation, and Management, 13th Edition Carlos Coronel and Steven Morris, Chapter Nine, page 467-468

5.5 The Logical Design

Logical design is the second stage in the database design process. The logical design goal is to design an enterprise-wide database that is based on a specific data model but independent of physical-level details. Logical design requires that all objects in the conceptual model be mapped to the specific constructs used by the selected database model. For example, the logical design for a relational DBMS includes the specifications for the relations (tables), relationships, and constraints (in other words, domain definitions, data validations, and security views). The logical design is generally performed in four steps, which are listed in Table 9.6.

TABLE 9.6

LOGICAL DESIGN STEPS

STEP	ACTIVITY
1	Map the conceptual model to logical model components.
2	Validate the logical model using normalization.
3	Validate the logical model integrity constraints.
4	Validate the logical model against user requirements.

Such steps, like most of the data-modelling process, are not necessarily performed sequentially, but in an iterative fashion. The following sections cover these steps in more detail.

The conceptual design, and the table columns must correspond to the attributes specified in the conceptual design. The outcome of this process is a list of relations, attributes, and relationships that will be the basis for the next step.

Map the Conceptual Model to the Logical Model

Recall that the second phase of the DBLC is database design, which comprises three stages: conceptual design, logical design, and physical design, plus the critical decision of DBMS selection. Conceptual design is the first stage in the database design process. The goal at this stage is to design a database that is independent of database software and physical details. The output of this process is a conceptual data model that describes the main data entities, attributes, relationships, and constraints of a given problem domain. This design is descriptive and narrative in form. In other words, it is generally composed of a graphical representation as well as textual descriptions of the main data elements, relationships, and constraints. In this stage, data modelling is used to create an abstract database structure that rep-resents real-world objects in the most realistic way possible. The conceptual model must embody a clear understanding of the business and its functional areas. At this level of abstraction, the type of hardware and database model to be used might not have been identified yet. Therefore, the design must be software-and hardware-independent so that the system can be set up within any platform chosen later.

Validate the Logical Model Using Normalization

The logical design should contain only properly normalized tables. The process of map-ping the conceptual model to the logical model may unveil some new attributes or the discovery of new multivalued or composite attributes. Therefore, it’s very likely that new attributes may be added to tables, or that entire new tables may be added to the logical model. For each identified table (old and new), you must ensure that all attributes are fully dependent on the identified primary key and that the tables are in at least third normal form (3NF).

As indicated throughout this section, database design is an iterative process. Activities such as normalization take place at different stages in the design process. Each time you reiterate a step, the model is further refined and better documented. New attributes may be created and assigned to the proper entities. Functional dependencies among determinant and dependent attributes are evaluated and data anomalies are prevented via normalization.

Validate Logical Model Integrity Constraints

The translation of the conceptual model into a logical model also requires definition of the attribute domains and appropriate constraints.

All the defined constraints must be supported by the logical data model. In this stage, you must map these constraints to the proper relational model constraints.

During this step, you also define which attributes are mandatory and which are optional and ensure that all entities maintain entity and referential integrity. The right to use the database is also specified during the logical design phase. Who will be allowed to use the tables, and what portions of the tables will be available to which users? Within a relational framework, the answers to those questions require the definition of appropriate views.

Special attention is needed at this stage to ensure that all views can be resolved, and that security is enforced to ensure the privacy of the data. Additionally, if you are working with a distributed database design, data could be stored at multiple locations, and each location may have different security restrictions. After validating the logical model integrity constraints, you are ready to validate the model against the end-user requirements. 9-6d validate the Logical model against user requirement

Validate the Logical Model against User Requirements

The logical design translates the software-independent conceptual model into a soft-ware-dependent model. The final step in the logical design process is to validate all logical model definitions against all end-user data, transaction, and security requirements. The stage is now set to define the physical requirements that allow the system to function within the selected DBMS/hardware environment.

Read

Database Systems: Design, Implementation, and Management, 13th Edition Carlos Coronel and Steven Morris, Chapter Nine, page 468-471.

5.6 The Physical Design

Physical design is the process of determining the data storage organization and data access characteristics of the database to ensure its integrity, security, and performance. This is the last

stage in the database design process. The storage characteristics are a function of the types of devices supported by the hardware, the type of data access methods supported by the system, and the DBMS. Physical design can become a very technical job that affects not only the accessibility of the data in the storage device(s) but the performance of the system. The physical design stage consists of the steps in Table 9.8

TABLE 9.8

PHYSICAL DESIGN STEPS

STEP	ACTIVITY
1	Define data storage organization.
2	Define integrity and security measures.
3	Determine performance measurements.

The following sections cover these steps in more detail.

Define Data Storage Organisation

Before you can define data storage organization, you must determine the volume of data to be managed and the data usage patterns.

- Knowing the data volume will help you determine how much storage space to reserve for the database. To do this, the designer follows a process similar to the one used during ER model verification. For each table, identify all possible transactions, their frequency, and volume. For each transaction, you determine the amount of data to be added or deleted from the database. This information will help you determine the amount of data to be stored in the related table.
- Conversely, knowing how frequently new data is inserted, updated, and retrieved will help the designer determine the data usage patterns. Usage patterns are critical, particularly in distributed database design. For example, are there any weekly batch uploads or monthly aggregation reports to be generated? How frequently is new data added to the system? Equipped with the two previous pieces of information, the designer must:
- Determine the location and physical storage organization for each table. As you saw in Section 9-3c, tables are stored in table spaces, and a table space can hold data from multiple tables. In this step, the designer assigns which tables will use which table spaces and assigns the location of the table spaces. For example, a useful technique available in most relational databases is the use of clustered tables. The clustered tables storage technique stores related rows from two related tables in adjacent data blocks on disk. This ensures that the data is stored in sequentially adjacent locations, thereby reducing data access time and increasing system performance.
- Identify indexes and the type of indexes to be used for each table. As you saw in previous chapters, indexes are useful for ensuring the uniqueness of data values in a column and to facilitate data lookups. You also know that the DBMS automatically creates a unique index for the primary key of each table. You will learn in Chapter 11 about the various types of index organization. In this step, you identify all required indexes and determine the best type of organization to use based on the data usage patterns and performance requirements.
- Identify the views and the type of views to be used on each table. As you learned in Chapter 8, a view is useful to limit access to data based on user or transaction needs. Views can also be used to simplify processing and end-user data access. In this step the designer must ensure that all views can be implemented and that they provide only the required data. The designer must also become familiar with the types of views supported by the DBMS and how they could help meet system goals.

Define Integrity and Security Measures

Once the physical organization of the tables, indexes, and views are defined, the database is ready for the end users. However, before users can access the data in the database, they must be properly authenticated. In this step of physical design, two tasks must be addressed:

- Define user and security groups and roles. User management is more a function of database administration than database design. However, as a designer you must know the different types of users and groups of users to properly enforce database security. Most DBMS implementations support the use of database roles. A database role is a set of database privileges that could be assigned as a unit to a user or group. For example, you could define an Advisor role that has Read access to the schedule view
- Assign security controls. The DBMS also allows administrators to assign specific access rights for database objects to a user or group of users. For example, you could assign the SELECT and UPDATE access rights to the user leaders on the CLASS table. An access right could also be revoked from a specific user or groups of users. This feature could come in handy during database backups, scheduled maintenance events, or even during data breach incidents.

Determine Performance Measures

Physical design becomes more complex when data is distributed at different locations because the performance is affected by the communication media's throughput. Given such complexities, it is not surprising that designers favour database software that hides as many of the physical-level activities as possible. Despite that relational models tend to hide the complexities of the computer's physical characteristics; the performance of relational databases is affected by physical storage properties. For example, performance can be affected by characteristics of the storage media, such as seek time, sector and block (page) size, buffer pool size, and the number of disk platters and read/write heads. In addition, factors such as the creation of an index can have a considerable effect on the relational database's performance—that is, data access speed and efficiency. In summary, physical design performance measurement deals with fine-tuning the DBMS and queries to ensure that they will meet end-user performance requirements.

Read

Database Systems: Design, Implementation, and Management, 13th Edition Carlos Coronel and Steven Morris, Chapter Nine, page 471-471.

5.7 Design Strategies

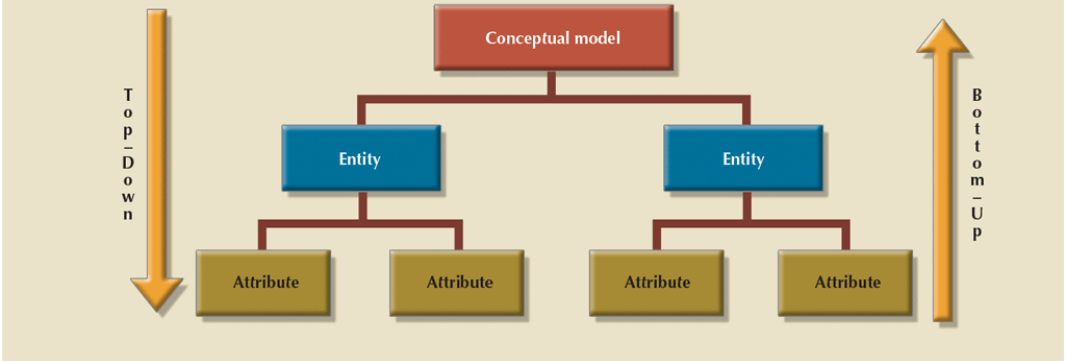
There are two classical approaches to database design:

- Top-down design starts by identifying the data sets and then defines the data elements for each of those sets. This process involves the identification of different entity types and the definition of each entity's attributes.

- Bottom-up design first identifies the data elements (items) and then groups them together in data sets. In other words, it first defines attributes, and then groups them to form entities.

The two approaches are illustrated in Figure 9.14. Selecting a primary emphasis on top-down or bottom-up procedures often depends on the scope of the problem or on personal preferences. Although the two methodologies are complementary rather than mutually exclusive, a primary emphasis on a bottom-up approach may be more productive for small databases with few entities, attributes, relations, and transactions. For situations in which the number, variety, and complexity of entities, relations, and transactions is overwhelming, a primarily top-down approach may be easier. Most companies have standards for systems development and database design already in place.

FIGURE 9.14 TOP-DOWN VS. BOTTOM-UP DESIGN SEQUENCING



Read

Database Systems: Design, Implementation, and Management, 13th Edition Carlos Coronel and Steven Morris, Chapter Nine, page 473-474.

5.8 Centralized Versus Decentralized Design

The two general approaches to database design (bottom-up and top-down) can be influenced by factors such as the scope and size of the system, the company's management style, and the company's structure (centralized or decentralized). Depending on these factors, the database design may be based on two very different design philosophies: centralized and decentralized. Centralized design is productive when the data component has a relatively small number of objects and procedures. The design can be carried out and represented in a fairly simple database. Centralized design is typical of relatively simple, small databases and can be successfully done by a single database administrator or by a small, informal design team. The company operations and the scope of the problem are sufficiently limited to allow even a single designer to define the problem(s), create the conceptual design, verify the conceptual design with the user views, define system processes and data constraints to ensure the efficacy of the design, and ensure that the design will

com-ply with all the requirements. (Although centralized design is typical for small companies, do not make the mistake of assuming that it is limited to small companies. Even large companies can operate within a relatively simple database environment.) Figure 9.15 summarizes the centralized design option. Note that a single conceptual design is completed and then validated in the centralized design approach. Decentralized design might be used when the system's data component has a considerable number of entities and complex relations on which very complex operations are performed. Decentralized design is also often used when the problem itself is spread across several operational sites and each element is a subset of the entire data set.

In large and complex projects, the database typically cannot be designed by only one person. Instead, a carefully selected team of database designers tackles a complex data-base project. Within the decentralized design framework, the database design task is divided into several modules. Once the design criteria have been established, the lead designer assigns design subsets or modules to design groups within the team.

Because each design group focuses on modelling a subset of the system, the definition of boundaries and the interrelation among data subsets must be very precise. Each design group creates a conceptual data model corresponding to the subset being modelled. Each conceptual model is then verified individually against the user views, processes, and constraints for each of the modules. After the verification process has been completed, all modules are integrated into one conceptual model. Because the data dictionary describes the characteristics of all objects within the conceptual data model, it plays a vital role in the integration process. After the subsets have been aggregated into a larger conceptual model, the lead designer must verify that it still can support all of the required transactions. Keep in mind that the aggregation process requires the designer to create a single model in which various aggregation problems must be addressed.

- Synonyms and homonyms. Various departments might know the same object by different names (synonyms), or they might use the same name to address different objects (homonyms). The object can be an entity, an attribute, or a relationship.
- Entity and entity subtypes. An entity subtype might be viewed as a separate entity by one or more departments. The designer must integrate such subtypes into a higher-level entity.
- Conflicting object definitions. Attributes can be recorded as different types (character, numeric), or different domains can be defined for the same attribute. Constraint definitions can vary as well. The designer must remove such conflicts from the model.
- An information system is designed to help transform data into information and to manage both data and information. Thus, the database is a very important part of the information system. Systems analysis is the process that establishes the need for an information system and its extent. Systems development is the process of creating an information system.
- The Systems Development Life Cycle (SDLC) traces the history of an application within the information system. The SDLC can be divided into five phases: planning, analysis, detailed systems design, implementation, and maintenance. The SDLC is an iterative process rather than a sequential process.
- The Database Life Cycle (DBLC) describes the history of the database within the information system. The DBLC is composed of six phases: database initial study, database design, implementation and loading, testing and evaluation, operation, and maintenance and evolution. Like the SDLC, the DBLC is iterative rather than sequential.
- The conceptual portion of the design may be subject to several variations based on two basic design philosophies: bottom-up versus top-down and centralized versus decentralized.

Read

Database Systems: Design, Implementation, and Management, 13th Edition Carlos Coronel and Steven Morris, Nine page 474-477.

Chapter Summary/Review

In this chapter, you have learned:

- An information system is designed to help transform data into information and to manage both data and information. Systems analysis is the process that establishes the need for an information system and its extent. Systems development is the process of creating an information system.

- The Systems Development Life Cycle (SDLC) traces the history of an application within the information system. The SDLC can be divided into five phases: planning, analysis, detailed systems design, implementation, and maintenance. The SDLC is an iterative process rather than a sequential process.

- The Database Life Cycle (DBLC) describes the history of the database within the information system. The DBLC is composed of six phases: database initial study, database design, implementation and loading, testing and evaluation, operation, and maintenance and evolution. Like the SDLC, the DBLC is iterative rather than sequential.

- The conceptual portion of the design may be subject to several variations based on two basic design philosophies: bottom-up versus top-down and centralized versus decentralized.

Review Questions

1. What is an information system? What is its purpose?

2. Discuss the distinction between centralized and decentralized conceptual database design.

3. What is the data dictionary's function in database design?

4. List and briefly explain the four steps performed during the logical design stage.

5. What does the acronym SDLC mean, and what are the five phases of the SDLC?

Problems

6. Suppose that you have been asked to create an information system for a manufacturing plant that produces nuts and bolts of many shapes, sizes, and functions. What questions would you ask, and how would the answers affect the database design?
    a. What do you envision the SDLC to be?
    b. What do you envision the DBLC to be?

7. In a construction company, a new system has been in place for a few months and now there is a list of possible changes/updates that need to be done. For each of the changes/updates, specify what type of maintenance needs to be done: (a) corrective, (b) adaptive, and (c) perfective.
    a. An error in the size of one of the fields has been identified and it needs to be updated status field needs to be changed.
    b. The company is expanding into a new type of service and this will require to enhancing the system with a new set of tables to support this new service and integrate it with the existing data.
    c. The company has to comply with some government regulations. To do this, it will require adding a couple of fields to the existing system tables.

8. You have been assigned to design the database for a new soccer club. Indicate the most appropriate sequence of activities by labelling each of the following steps in the correct order. (e.g., if you think that "Load the database" is the appropriate first step, label it "1.") Create the application programs.
    _____ Create a description of each system process.
    _____ Test the system. Load the database.
    _____ Normalize the conceptual model.
    _____ Interview the soccer club president.
    _____ Create a conceptual model using ER diagrams.
    _____ Interview the soccer club director of coaching.
    _____ Create the file (table) structures.
    _____ Obtain a general description of the soccer club operations.
    _____ Draw a data flow diagram and system flowcharts

Review Questions (MCQ)

1. A conceptual design

    a. Is a documentation technique
    b. Needs data volume and processing frequencies to determine the size of the database
    c. Involves modelling independent of the DBMS
    d. Is designing relational model

2. The logical design of a database is called a

    a. Database instance
    b. Database snapshot
    c. Database schema
    d. All of the above

3. _____ is a classical approach to database design

    a. Bottom-up approach
    b. Top-down approach
    c. Left-right approach
    d. Right-left approach
    e. None of the above

4. This phase of the SDLC is known as the “ongoing phase” where the system is periodically evaluated and updated as needed.

    a. Preliminary investigation
    b. System design
    c. System implementation
    d. System maintenance

5. The final step of the system analysis phase in the SDLC is to

    a. Gather data
    b. Write system analysis report
    c. Propose changes
    d. Analyse data

LEARNING OUTCOMES

After reading this Section of the guide, the learner should be able to:

- Identify the procedures involved in database performance tuning
- Describe how a DBMS processes SQL queries in each of its three phases
- Explain the role of indexes in speeding up data access
- Differentiate between a rule-based optimizer and a cost-based optimizer
- Describe some common practices used to write efficient SQL Code
- Explain how to formulate queries and tune the DBMS for optimal performance

6.1 Database Performance Tuning Concepts

One of the main functions of a database system is to provide timely answers to end users. End users interact with the DBMS through queries to generate information, using the following sequence:

1. The end-user (client-end) application generates a query.
2. The query is sent to the DBMS (server end).
3. The DBMS (server end) executes the query.
4. The DBMS sends the resulting data set to the end-user (client-end) application.

End users expect their queries to return results as quickly as possible. Good database performance is hard to evaluate but it is easier to identify bad database performance than good database performance—all it takes is end-user complaints about slow query results. Regardless of end-user perceptions, the goal of database performance is to execute queries as fast as possible. Therefore, database performance must be closely monitored and regularly tuned. Database performance tuning refers to a set of activities and procedures designed to reduce the response time of the database system.

In general, the performance of a typical DBMS is constrained by three main factors: CPU processing power, available primary memory (RAM), and input/output (hard disk and network) throughput. Table 11.1 lists some system components and summarizes general guidelines for achieving better query performance.

Fine-tuning the performance of a system requires a holistic approach. With that said, good database performance starts with good database design. No amount of fine-tuning will make a poorly designed database perform as well as expected.

What constitutes a good, efficient database design? From the performance-tuning point of view, the database designer must ensure that the design makes use of features in the DBMS that guarantee the integrity and optimal performance of the database. This chapter provides fundamental knowledge that will help you optimize database performance by selecting the appropriate database server configuration, using indexes, under-standing table storage organization and data locations, and implementing the most efficient SQL query syntax.

Performance Tuning: Client and Server

In general, database performance-tuning activities can be divided into those on the client side and those on the server side.

- On the client side, the objective is to generate a SQL query that returns the correct answer in the least amount of time, using the minimum amount of resources at the server end. The activities required to achieve that goal are commonly referred to as **SQL performance tuning.**
- On the server side, the DBMS environment must be properly configured to respond to clients' requests in the fastest way possible, while making optimum use of existing resources. The activities required to achieve that goal are commonly referred to as **DBMS performance tuning**

In this chapter, we assume a fully optimized network, and, therefore, our focus is on the database components. Even in multi-tier client/server environments that consist of a client front end, application middleware, and database server back end, performance-tuning activities are frequently

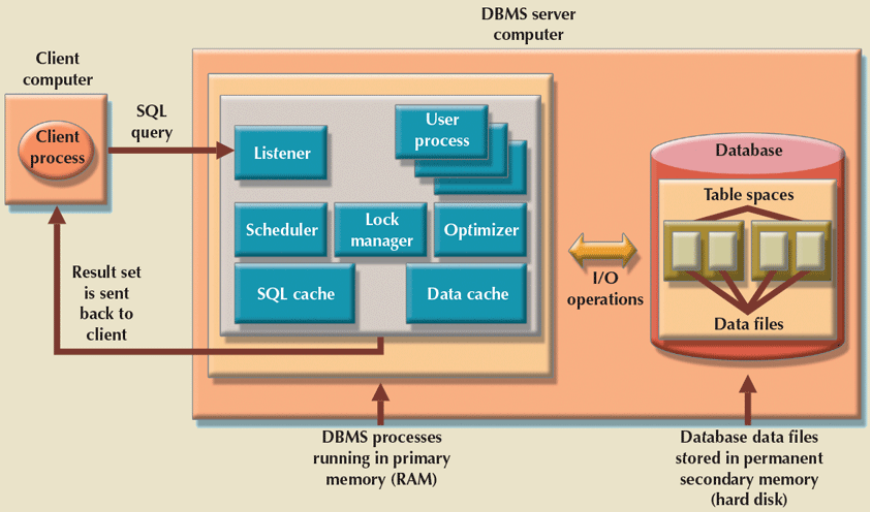
divided into subtasks to ensure the fastest possible response time between any two component points.

The database administrator must work closely with the network group to ensure that database traffic flows efficiently in the network infrastructure. This is even more important when you consider that most database systems service geographically dispersed users. This chapter covers SQL performance-tuning practices on the client side and DBMS performance-tuning practices on the server side. However, before you start learning about the tuning processes, you must first learn more about the DBMS architectural components and processes, and how those processes interact to respond to end-users' requests.

DBMS Architecture

The architecture of a DBMS is represented by the processes and structures (in memory and permanent storage) used to manage a database. Such processes collaborate with one another to perform specific functions. Figure 11.1 illustrates the basic DBMS architecture.

FIGURE 11.1 BASIC DBMS ARCHITECTURE



Database Query Optimization Modes

Most of the algorithms proposed for query optimization are based on two principles:

- The selection of the optimum execution order to achieve the fastest execution time
- The selection of sites to be accessed to minimize communication costs

Within those two principles, a query optimization algorithm can be evaluated on the basis of its operation mode or the timing of its optimization.

Operation modes can be classified as manual or automatic. Automatic query optimization means that the DBMS finds the most cost-effective access path with-out user intervention and is thus more desirable. manual query optimization requires that the optimization be selected and scheduled by the end user or programmer.

Query optimization algorithms can also be classified according to when the optimization is done. Within this timing classification, query optimization algorithms can be static or dynamic.

- Static query optimization takes place at compilation time. In other words, the best optimization strategy is selected when the query is compiled by the DBMS.
- Dynamic query optimization takes place at execution time. Database access strategy is defined when the program is executed. Therefore, access strategy is dynamically determined by the DBMS at run time, using the most up-to-date information about the database.

Finally, query optimization techniques can be classified according to the type of information that is used to optimize the query. For example, queries may be based on statistically based or rule-based algorithms.

- A statistically based query optimization algorithm uses statistical information, such as size, number of records, average access time and such like information about the database. These statistics are then used by the DBMS to determine the best access strategy.
- The statistical information is managed by the DBMS and is generated in one of two different modes: dynamic or manual. In the dynamic statistical generation mode, the DBMS automatically evaluates and updates the statistics after each data access operation. In the manual statistical generation mode, the statistics must be updated periodically through a user-selected utility such as IBM's RUNSTAT com-mand, which is used by DB2 DBMSs.
- A rule-based query optimization algorithm is based on a set of user-defined rules to determine the best query access strategy. The rules are entered by the end user or database administrator, and they are typically general in nature.

Because database statistics play a crucial role in query optimization, this topic is explored in more detail in the next section.

Read

Database Systems: Design, Implementation, and Management, 13th Edition Carlos Coronel and Steven Morris, Chapter Eleven page 516-522.

6.2 Query Processing and Optimization

What happens at the DBMS server end when the client's SQL statement is received? In simple terms, the DBMS processes a query in three phases: parsing, execution and fetching.

SQL Parsing Phase

The optimization process includes breaking down—parsing—the query into smaller units and transforming the original SQL query into a slightly different version of the original SQL code, but one that is fully equivalent and more efficient. Fully equivalent means that the optimized query results are always the same as the original query. More efficient means that the optimized query

will almost always execute faster than the original query. To determine the most efficient way to execute the query, the DBMS may use the database statistics you learned about earlier.

The SQL parsing activities are performed by the query optimizer, which analyses the SQL query and finds the most efficient way to access the data. This process is the most time-consuming phase in query processing. Parsing a SQL query requires several steps, in which the SQL query is:

- Validated for syntax compliance
- Validated against the data dictionary to ensure that table names and column names are correct
- Validated against the data dictionary to ensure that the user has proper access rights
- Analysed and decomposed into more atomic components
- Optimized through transformation into a fully equivalent but more efficient SQL query
- Prepared for execution by determining the most efficient execution or access plan

Access plans are DBMS-specific and translate the client's SQL query into the series of complex I/O operations required to read the data from the physical data files and generate the result set.

SQL Execution Phase

In this phase, all I/O operations indicated in the access plan are executed. When the execution plan is run, the proper locks—if needed—are acquired for the data to be accessed, and the data is retrieved from the data files and placed in the DBMS's data cache. All transaction management commands are processed during the parsing and execution phases of query processing.

SQL Fetching Phase

After the parsing and execution phases are completed, all rows that match the specified condition(s) are retrieved, sorted, grouped, and aggregated (if required). During the fetching phase, the rows of the resulting query result set are returned to the client. In this stage, the database server coordinates the movement of the result set rows from the server cache to the client cache.

Query Processing Bottlenecks

As you have seen, the execution of a query requires the DBMS to break down the query into a series of interdependent I/O operations to be executed in a collaborative manner. The more complex a query is, the more complex the operations are, which means that bottlenecks are more likely.

A **query processing bottleneck** is a delay introduced in the processing of an I/O operation that causes the overall system to slow down. Within a DBMS, five components typically cause bottlenecks:

- RAM.
- Hard disk.
- Network.
- Application code.

Bottlenecks are the result of multiple database transactions competing for the use of database resources (CPU, RAM, hard disk, indexes, locks, buffers, etc.). Because most (if not all) transactions work with data rows in tables, one of the most typical bottlenecks is caused by transactions competing for the same data rows. Another common source of contention is for shared

memory resources, particularly shared buffers and locks. To speed up data update operations, the DMBS uses buffers to cache the data. At the same time, to manage access to data, the DBMS uses locks. Learning how to avoid these bottlenecks and optimize database performance is the focal point of this chapter.

Read

Database Systems: Design, Implementation, and Management, 13th Edition Carlos Coronel and Steven Morris, Chapter Eleven, page 522-526

Indexes and Query Optimization

Indexes are crucial in speeding up data access because they facilitate searching, sorting, and using aggregate functions and even join operations. The improvement in data access speed occurs because an index is an ordered set of values that contains the index key and pointers. The pointers are the row IDs for the actual table rows. Conceptually, a data index is like a book index.

An index scan is more efficient than a full table scan because the index data is pre-ordered, and the amount of data is usually much smaller. Therefore, when performing searches, it is almost always better for the DBMS to use the index to access a table than to scan all rows in a table sequentially.

Given a table with 14,786 customer records, in the absence of an index, the DBMS will perform a full-table scan and read all 14,786 customer rows. Assuming an index is created (and analysed), the DBMS would be saved from reading approximately 14,776 I/O requests for customer rows that do not meet the criteria. That is a lot of CPU cycles!

If indexes are so important, why not index every column in every table? The simple answer is that it is not practical to do so. Indexing every column in every table overtaxes the DBMS in terms of index-maintenance processing, especially if the table has many attributes and rows, or requires many inserts, updates, and deletes.

One measure that determines the need for an index is the data sparsity of the column you want to index. **Data sparsity** refers to the number of different values a column could have. Knowing the sparsity helps you decide whether the use of an index is appropriate. In subsequent sections you will learn and be able to identify when an index is necessary.

Most DBMSs implement indexes using one of the following data structures:

- Hash index. A hash index is based on an ordered list of hash values, created using a hash algorithm from a key column.
- B-tree index. The B-tree index is an ordered data structure organized as an upside-down tree.
- Bitmap index. A bitmap index uses a bit array (0s and 1s) to represent the existence of a value or condition.

Using the preceding index characteristics, a database designer can determine the best type of index to use. Because the REGION_CODE column contains only a few different values that repeat a relatively large number of times compared to the total number of rows in the table, a bitmap index

will be used. Figure 11.4 shows the B-tree and bitmap representations for the CUSTOMER table used in the previous discussion.

Current-generation DBMSs are intelligent enough to determine the best type of index to use under certain circumstances, provided that the DBMS has updated data-base statistics. Regardless of which index is chosen, the DBMS determines the best plan to execute a given query. The next section guides you through a simplified example of the type of choices the query optimizer must make.

Read

Database Systems: Design, Implementation, and Management, 13th Edition Carlos Coronel and Steven Morris, Chapter Eleven, page 526-528

Optimizer Choices

Using Hints to Affect Optimizer Choices

Query optimization is the central activity during the parsing phase in query processing. In this phase, the DBMS must choose what indexes to use, how to perform join operations, which table to use first, and so on. Each DBMS has its own algorithms for determining the most efficient way to access the data. The query optimizer can operate in one of two modes:

- A **rule-based optimizer** uses pre-set rules and points to determine the best approach to execute a query.
- A **cost-based optimizer** uses sophisticated algorithms based on statistics about the objects being accessed to determine the best approach to execute a query.

In this case, the optimizer process adds up the processing cost, the I/O costs, and the resource costs (RAM and temporary space) to determine the total cost of a given execution plan.

The optimizer's objective is to find alternative ways to execute a query—to evaluate the "cost" of each alternative and then to choose the one with the lowest cost. To understand the function of the query optimizer, consider a simple example. Assume that you want to list all products provided by a vendor based in Florida. To acquire that information, you could write the following query:

```
SELECT    P_CODE, P_DESCRIPT, P_PRICE, V_NAME, V_STATE
FROM      PRODUCT, VENDOR
WHERE     PRODUCT.V_CODE = VENDOR.V_CODE AND VENDOR.V_STATE = 'FL';
```

Furthermore, assume that the database statistics indicate the following:

- The PRODUCT table has 7,000 rows.
- The VENDOR table has 300 rows.
- Ten vendors are in Florida.
- 1000 products come from vendors in Florida.

It is important to point out that only the first two items are available to the optimizer. The second two items are assumed to illustrate the choices that the optimizer must make. Armed with the information in the first two items, the optimizer would try to find the most efficient way to access the data. The primary factor in determining the most efficient access plan is the I/O cost. (Remember, the DBMS always tries to minimize I/O operations.) Table 11.4 shows two sample access plans (Plan A and Plan B) for the previous query and their respective I/O costs.

TABLE 11.4

COMPARING ACCESS PLANS AND I/O COSTS

PLAN	STEP	OPERATION	I/O OPERATIONS	I/O COST	RESULTING SET ROWS	TOTAL I/O COST
A	A1	Cartesian product (PRODUCT, VENDOR)	7,000 + 300	7,300	2,100,000	7,300
	A2	Select rows in A1 with matching vendor codes	2,100,000	2,100,000	7,000	2,107,300
	A3	Select rows in A2 with V_STATE = 'FL'	7,000	7,000	1,000	**2,114,300**
B	B1	Select rows in VENDOR with V_STATE = 'FL'	300	300	10	300
	B2	Cartesian Product (PRODUCT, B1)	7,000 + 10	7,010	70,000	7,310
	B3	Select rows in B2 with matching vendor codes	70,000	70,000	1,000	**77,310**

Comparing, plan A and B, the optimizer would consider the number of I/O operations as well as the resulting data set and not only the total I/O cost, to decide which of the plans is optimal (best) and execute the query using that access plan. In analysing Table 11.4, you can see how Plan A has a total I/O cost that is almost 30 times higher than Plan B. In this case, the optimizer will choose Plan B to execute the SQL.

Read

Database Systems: Design, Implementation, and Management, 13th Edition Carlos Coronel and Steven Morris, Chapter Eleven, page 528-531

6.3 Performance Tuning

SQL performance tuning is evaluated from the client perspective. Therefore, the goal is to illustrate some common practices used to write efficient SQL code.

In as much as most current-generation relational DBMSs perform automatic query optimization at the server end and most SQL performance optimization techniques are DBMS-specific and, therefore, are rarely portable, even across different versions of the same DBMS. There is considerable room for improvement.

A poorly written SQL query can, and usually will, bring the database system to its knees from a performance point of view. Therefore, although a DBMS provides general optimizing services, a carefully written query almost always outperforms a poorly written one.

Although SQL data manipulation statements include many different commands such as INSERT, UPDATE, DELETE, and SELECT, most recommendations in this section are related to the use of the SELECT statement, and the use of indexes and how to write conditional expressions.

Index Selectivity

Indexes are the most important technique used in SQL performance optimization. The key is to know when an index is used. As a rule, indexes are likely to be used:

- When an indexed column appears by itself in the search criteria of a WHERE or HAVING clause
- When an indexed column appears by itself in a GROUP BY or ORDER BY clause
- When a MAX or MIN function is applied to an indexed column
- When the data sparsity on the indexed column is high

Indexes are very useful when you want to select a small subset of rows from a large table based on a given condition. If an index exists for the column used in the selection, the DBMS may choose to use it. The objective is to create indexes with high selectivity. **Index selectivity** is a measure of the likelihood that an index will be used in query processing. Here are some general guidelines for creating and using indexes:

- Create indexes for each single attribute used in a WHERE, HAVING, ORDER BY, or GROUP BY clause.
- Do not use indexes in small tables or tables with low sparsity.
- Declare primary and foreign keys so the optimizer can use the indexes in join operations.
- Declare indexes in join columns other than PK or FK.

You cannot always use an index to improve performance. It is so, because, in some DBMSs, indexes are ignored when you use functions in the table attributes. However, major data-bases such as Oracle, SQL Server, and DB2 now support function-based indexes. **A function-based index** is an index based on a specific SQL function or expression.

How many indexes should you create? It’s worth repeating that you should not create an index for every column in a table as it will slow down data manipulation operations.. Furthermore, some query optimizers will choose only one index to be the driving index for a query, even if your query uses conditions in many different indexed columns. Which index does the optimizer use? If you use the cost-based optimizer, the answer will change with time as new rows are added to or deleted from the tables. In any case, you should create indexes in all search columns and then let the optimizer choose. It is important to constantly evaluate the index usage—monitor, test, evaluate, and improve it if performance is not adequate.

Conditional Expressions

A conditional expression is normally placed within the WHERE or HAVING clauses of a SQL statement. Also known as conditional criteria, a conditional expression restricts the output of a query to only the rows that match the conditional criteria. Generally, the conditional criteria have the form shown in Table 11.6.

TABLE 11.6

CONDITIONAL CRITERIA

OPERAND1	CONDITIONAL OPERATOR	OPERAND2
P_PRICE	>	10.00
V_STATE	=	FL
V_CONTACT	LIKE	Smith%
P_QOH	>	P_MIN * 1.10

Most of the query optimization techniques mentioned below are designed to make the optimizer's work easier. The following common practices are used to write efficient conditional expressions in SQL code.

- Use simple columns or literals as operands in a conditional expression—avoid the use of conditional expressions with functions whenever possible. Comparing the contents of a single column to a literal is faster than comparing to expressions.
- Numeric field comparisons are faster than character, date, and NULL comparisons. In general, the CPU handles numeric comparisons (integer and decimal) faster.
- Equality comparisons are generally faster than inequality comparisons.
- Whenever possible, transform conditional expressions to use literals.
- When using multiple conditional expressions, write the equality conditions first.
- If you use multiple AND conditions, write the condition most likely to be false first. If you use this technique, the DBMS will stop evaluating the rest of the conditions as soon as it finds a conditional expression that is evaluated as false.
- When using multiple OR conditions, put the condition most likely to be true first. By doing this, the DBMS will stop evaluating the remaining conditions as soon as it finds a conditional expression that is evaluated as true.
- Whenever possible, try to avoid the use of the NOT logical operator.

Read

Database Systems: Design, Implementation, and Management, 13th Edition Carlos Coronel and Steven Morris, Chapter Eleven page 531-534.

Query Formulation

Queries are usually written to answer questions. Often, an end user will give you a sample output and tell you to match that output format, you must write the corresponding SQL. This requires that you must have a good understanding of the database environment and the database that will be the focus of your SQL code.

This section focuses on SELECT queries because they are the queries you will find in most applications. To formulate a query, you would normally follow these steps:

1. Identify what columns and computations are required.
   a. Do you need simple expressions?
   b. Do you need aggregate functions? In some cases, you might need to use a subquery.
   c. Determine the granularity of the raw data required for your output.
2. Identify the source tables.
3. Determine how to join the tables.
4. Determine what selection criteria are needed. Most queries involve some type of selection/comparison criteria, such as:
   a. Simple comparison.
   b. Single value to multiple values.
   c. Nested comparisons.
   d. Grouped data selection.
5. Determine the order in which to display the output.

Read

Database Systems: Design, Implementation, and Management, 13th Edition Carlos Coronel and Steven Morris, Chapter Eleven page 534-536.

6.4 DBMS Performance Tuning

DBMS performance tuning includes global tasks such as managing the DBMS processes in primary memory (allocating memory for caching purposes) and managing the structures in physical storage (allocating space for the data files).

Fine-tuning the performance of the DBMS also includes applying several practices examined in the previous section. DBMS performance tuning at the server end focuses on setting the parameters used for:

- Data cache. The data cache size must be set large enough to permit as many data requests as possible to be serviced from the cache.
- SQL cache. The SQL cache stores the most recently executed SQL statements (after the SQL statements have been parsed by the optimizer).
- Sort cache. The sort cache is used as a temporary storage area for ORDER BY or GROUP BY operations, as well as for index-creation functions.
- Optimizer mode.

From the performance point of view, it would be optimal to have the entire database stored in primary memory to minimize costly disk access. **In-memory database** systems are optimized to store large portions (if not all) of the database in primary (RAM) storage rather than secondary (disk) storage. These systems are becoming popular because increasing performance demands of

modern database applications (such as Big Data), diminishing costs, and technology advances of components.

Although in-memory databases are carving a niche in selected markets, most data-base implementations still rely on data stored on disk drives. That is why managing the physical storage details of the data files plays an important role in DBMS performance tuning. Note the following general recommendations for physical storage of databases:

- Use **I/O accelerators**. This type of device uses flash solid state drives (SSDs) to store the database.
- Use **RAID** (Redundant Array of Independent Disks) to provide both performance improvement and fault tolerance, and a balance between them. Fault tolerance means that in case of failure, data can be reconstructed and retrieved.
- Minimize disk contention. Use multiple, independent storage volumes with independent spindles (rotating disks) to minimize hard disk cycles. A database should have at least the following table spaces:
    - *System table space*. This is used to store the data dictionary tables.
    - *User data table space*. This is used to store end-user data.
    - *Index table space*. This is used to store indexes.
    - *Temporary table space*. This is used as a temporary storage area for merge, sort, or set aggregate operations.
    - *Rollback segment table space*. This is used for transaction-recovery purposes.
- Put high-usage tables in their own table spaces so the database minimizes conflict with other tables.
- Assign separate data files in separate storage volumes for the indexes, system, and high-usage tables. This ensures that index operations will not conflict with end-user data or data dictionary table access operations.
- Take advantage of the various table storage organizations available in the database.
- Partition tables based on usage.
- Use denormalized tables where appropriate.
- Store computed and aggregate attributes in tables. In short, use derived attributes in your tables.

Chapter Summary/Review

In this chapter, you have learned:

- Database performance tuning refers to a set of activities and procedures designed to ensure that an end-user query is processed by the DBMS in the least amount of time.
- Database statistics refer to a number of measurements gathered by the DBMS that describe a snapshot of the database objects' characteristics. The DBMS gathers statistics about objects such as tables, indexes, and available resources, and uses the statistics to make critical decisions about improving query processing efficiency.
- DBMSs process queries in three phases. In the parsing phase, the DBMS parses the SQL query and chooses the most efficient access/execution plan. In the execution phase, the DBMS executes the SQL query using the chosen execution plan. In the fetching phase, the DBMS fetches the data and sends the result set back to the client.
- Indexes are crucial in the process that speeds up data access. Indexes facilitate searching, sorting, and using aggregate functions and join operations.
- During query optimization, the DBMS must choose what indexes to use, how to perform join operations, which table to use first, and so on. Each DBMS has its own algorithms for determining the most efficient way to access the data. The two most common approaches are rule-based and cost-based optimization.
- A rule-based optimizer uses pre-set rules and points to determine the best approach to execute a query. A cost-based optimizer uses sophisticated algorithms based on statistics about the objects being accessed to determine the best approach to execute a query.
- SQL performance tuning deals with writing queries that make good use of the statistics. In particular, queries should make good use of indexes.
- Query formulation deals with how to translate business questions into specific SQL code to generate the required results. To do this, you must carefully evaluate which columns, tables, and computations are required to generate the desired output.
- DBMS performance tuning includes tasks such as managing the DBMS processes in primary memory (allocating memory for caching purposes) and managing the structures in physical storage (allocating space for the data files).

Review Questions

1. What is database performance tuning?
2. What database statistics measurements are typical of tables, indexes, and resources?
3. In simple terms, the DBMS processes a query in three phases. What are the phases, and what is accomplished in each phase?
4. What is the difference between a rule-based optimizer and a cost-based optimizer?
5. Give 3 general guidelines for creating and using indexes?

Problems

6. What recommendations would you make for managing the data files in a DBMS with many tables and indexes?

Problems 7–9 are based on the following query:

```
SELECT    EMP_LNAME, EMP_FNAME, EMP_DOB, YEAR(EMP_DOB) AS YEAR
FROM      EMPLOYEE
WHERE     YEAR(EMP_DOB) = 1976;
```

7. What is the likely data sparsity of the EMP_DOB column?
8. Should you create an index on EMP_DOB? Why or why not?
9. What type of database I/O operations will likely be used by the query?

Review Questions (MCQ)

1. Which of the following statements is correct?

    a. Avoid less row count SQL statements that can cause a table lock
    b. A blocking lock occurs when one lock causes another process to wait in a holding queue
    c. Lock is a done by database when any connection access a different piece of data concurrently
    d. None of the above

2. _____ allow concurrent transactions to read (SELECT) a resource.

a. Update locks
b. Shared locks
c. Exclusive Locks
d. All of the above

3. _____ rolls back a user-specified transaction to the beginning of the transaction.

a. ROLLBACK
b. ROLLBACK WORK
c. SAVE TRANSACTION
d. COMMIT

4. DBMSs process queries in three phases

a. Fetching phase, execution phase, and sending phase
b. Parsing phase, execution phase, and fetching phase
c. Forward phase, middle phase, and fetching phase
d. Parsing phase, implementing phase, and fetching phase

5. _____ deals with how to translate business questions into specific SQL code to generate the required results.

a. Query optimization
b. SQL performance tuning
c. Query formulation
d. DBMS performance tuning

LEARNING OUTCOMES

After reading this Section of the guide, the learner should be able to:

- Explain the purpose of standard database connectivity interfaces
- Describe the functionality and features of various database connectivity technologies: ODBC, OLE, ADO.NET, and JDBC
- Describe how web-to-database middleware is used to integrate databases with the internet
- Identify the services provides by the web application servers
- Describe the advantages and disadvantages of using cloud computing to database-as-a-service model

7.1 Database Connectivity

Database connectivity refers to the mechanisms through which application programs connect and communicate with data repositories. Before learning about the various data connectivity options, it is important to review some important fundamentals you have learned in this book:

- DBMSs provide means to interact with the data in their databases.
- Modern DBMSs have the option to store data locally or distributed in multiple locations.
- The database connectivity software we discuss in this chapter supports Structured Query Language (SQL) as the standard data manipulation language.
- Database connectivity software works in a client/server architecture, by which processing tasks are split among multiple software layers.

In the case of database connectivity software, also known as database middleware, you could break down its basic functionality into three broad layers:

1. A data layer where the data resides.
2. A middle layer that manages multiple connectivity and data transformation issues.
3. A top layer that interfaces with the actual external application.

Database connectivity software provides an interface between the application program and the database or data repository. The data repository, also known as the data source, represents the data management application, such as Oracle, SQL Server, IBM DB2, or NoSQL that will be used to store the data generated by the application program.

Ideally, a data source or data repository could be located anywhere and hold any type of data. Furthermore, the same database connectivity middleware could support multiple data sources at the same time. This multi-data-source type capability is based on the support of well-established data access standards. The need for standard database connectivity interfaces cannot be overstated. Although there are many ways to achieve database connectivity, this section covers only the following interfaces:

- Native SQL connectivity (vendor provided)
- Microsoft's Open Database Connectivity (ODBC), Data Access Objects (DAO), and Remote Data Objects (RDO)
- Microsoft's Object Linking and Embedding for Database (OLE-DB)
- Microsoft's ActiveX Data Objects (ADO.NET)
- Oracle's Java Database Connectivity (JDBC)

The data connectivity interfaces illustrated here are dominant players in the market, and more importantly, they enjoy the support of most database vendors. In fact, ODBC, OLE-DB, and ADO.NET form the backbone of Microsoft's Universal Data Access (UDA) architecture, a collection of technologies used to access any type of data source and manage the data through a common interface. As you will see, Microsoft's database connectivity interfaces have evolved over time: each interface builds on top of the other, thus providing enhanced functionality, features, flexibility, and support.

Native SQL Connectivity

Most DBMS vendors provide their own methods for connecting to their databases. Native SQL connectivity refers to the connection interface that is provided by the database vendor and is unique to that vendor. The best example of this type of native interface is the Oracle RDBMS. To connect

a client application to an Oracle database, you must install and configure Oracle's SQL*Net interface on the client computer.

Native database connectivity interfaces are optimized for "their" DBMS, and those interfaces support access to most or all the database features. However, maintaining multiple native interfaces for different databases can become a burden for the program-mer. Therefore, the need for universal database connectivity arises. Usually, the native database connectivity interface provided by the vendor is not the only way to connect to a database; most current DBMS products support other database connectivity standards, the most common being ODBC.

ODBC, DAO, and RDO

Developed in the early 1990s, Open Database Connectivity (ODBC) is Microsoft's implementation of a superset of the SQL Access Group Call Level Interface (CLI) standard for database access. ODBC allows any Windows application to access relational data sources, using SQL via a standard application programming interface (API).

As programming languages evolved, ODBC did not provide significant functionality beyond the ability to execute SQL to manipulate relational-style data. To counter-act that, Microsoft developed two other data access interfaces:

- Data Access Objects (DAO) is an object-oriented API used to access desktop databases.
- Remote Data Objects (RDO) is a higher-level, object-oriented application interface used to access remote database servers.

The basic ODBC architecture has three main components:

- A high-level ODBC API through which application programs access ODBC functionality
- A driver manager that is managing all database connections
- An ODBC driver that communicates directly to the DBMS

Defining a data source is the first step in using ODBC. To define a data source, you must create a **data source name (DSN)** for it. To create a DSN, you need to provide the following:

- An ODBC driver. You must identify the driver to use to connect to the data source.
- A name. This is a unique name by which the data source will be known to ODBC, and therefore to applications.
- ODBC driver parameters. Most ODBC drivers require specific parameters to establish a connection to the database.

Once the ODBC data source is defined, application programmers can write to the ODBC API by issuing specific commands and providing the required parameters. The ODBC Driver Manager will properly route the calls to the appropriate data source.

The ODBC API standard defines three levels of compliance: Core, Level-1, and Level-2, which provide increasing levels of functionality. Because much of the functionality provided by these interfaces is oriented toward accessing relational data sources, the use of the interfaces was limited with other data source types. With the advent of object-oriented programming languages, it has become more important to provide access to other nonrelational data sources.

OLE-DB and ADO

ODBC, DAO, and RDO do not provide support for nonrelational data. To answer that need and to simplify data connectivity, Microsoft developed Object Linking and Embedding for Database (OLE-DB). Based on Microsoft's Component Object Model (COM), OLE-DB is database middleware that adds object-oriented functionality for access to relational and nonrelational data.

OLE-DB is composed of a series of COM objects that provide low-level database connectivity for applications. The OLE-DB model is better understood when you divide its functionality into two types of objects:

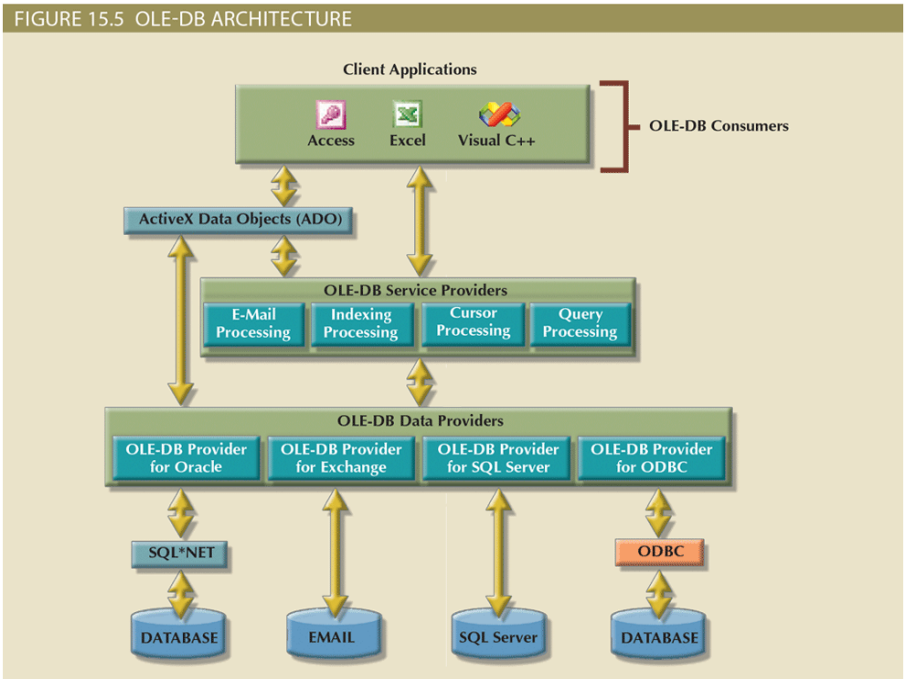
- *Consumers* are objects (applications or processes) that request and use data.
- *Providers* are objects that manage the connection with a data source and provide data to the consumers. Providers are divided into two categories: data providers and service providers.

As a common practice, many vendors provide OLE-DB objects to augment their ODBC support, effectively creating a shared object layer on top of their existing database connectivity (ODBC or native) through which applications can interact. The OLE-DB objects expose functionality about the database.

Additionally, the objects implement specific tasks, such as establishing a connection, executing a query, invoking a stored procedure, defining a transaction, or invoking an OLAP function. By using OLE-DB objects, the database vendor can choose what functionality to implement in a modular way, instead of being forced to include all the functionality all the time.

OLE-DB provides additional capabilities for the applications accessing the data. However, it does not provide support for scripting languages, especially the ones used for web development. To provide that support, Microsoft developed a new object framework called ActiveX Data Objects (ADO), which provides a high-level, application-oriented interface to interact with OLE-DB, DAO, and RDO. ADO provides a unified interface to access data from any programming language that uses the underlying OLE-DB objects. Figure 15.5 illustrates the ADO/OLE-DB architecture and how it interacts with ODBC and native connectivity options.

FIGURE 15.5 OLE-DB ARCHITECTURE



ADO introduced a simpler object model that was composed of only a few interacting objects to provide the data manipulation services required by the applications. Although the ADO model is a

tremendous improvement over the OLE-DB model, Microsoft is actively encouraging programmers to use its newer data access framework, ADO.NET.

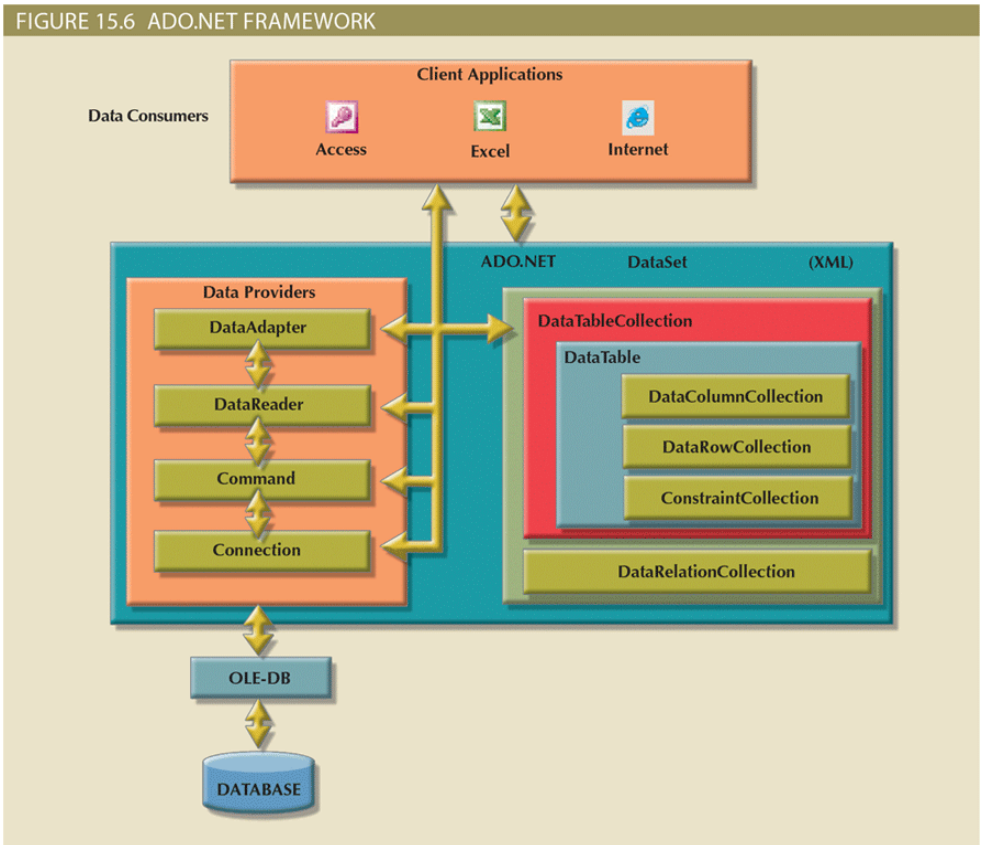
ADO.NET

Based on ADO, ADO.NET is the data access component of Microsoft's .NET application development framework. The Microsoft .NET framework is a component-based platform for developing distributed, heterogeneous, interoperable applications aimed at manipulating any type of data using any combination of network, operating system, and programming language. Comprehensive coverage of the .NET framework is beyond the scope of this book. Therefore, this section only introduces the basic data access component of the .NET architecture, ADO.NET.

It is important to understand that the .NET framework extends and enhances the functionality provided by the ADO/OLE-DB duo. ADO.NET introduced two new features that are critical for the development of distributed applications: DataSets and XML support.

The ADO.NET framework consolidates all data access functionality under one integrated object model. In this object model, several objects interact with one another to perform specific data manipulations. These objects can be grouped as data providers and consumers. Data provider objects are provided by the database vendors. However, ADO.NET comes with two standard data providers: one for OLE-DB data sources and one for SQL Server. ADO.NET includes a highly optimized data provider for SQL Server. Whatever the data provider is, it must support a set of specific objects to manipulate the data in the data source. Some of those objects are shown in Figure 15.6.

FIGURE 15.6 ADO.NET FRAMEWORK



The ADO.NET framework is optimized to work in disconnected environments. In a disconnected environment, applications exchange messages in request/reply format. The most common example

of a disconnected system is the Internet. Modern applications rely on the Internet as the network platform and on the web browser as the graphical user interface. In later sections, you will learn about how Internet databases work.

Java Database Connectivity (JDBC)

Java is an object-oriented programming language developed by Sun Microsystems (acquired by Oracle in 2010) that runs on top of web browser software. When Java applications need to access data outside the Java runtime environment, they use predefined application programming interfaces. **Java Database Connectivity (JDBC)** is an application programming interface that allows a Java program to interact with a wide range of data sources, including relational databases, tabular data sources, spreadsheets, and text files. JDBC allows a Java program to establish a connection with a data source, prepare and send the SQL code to the database server, and process the result set.

One main advantage of JDBC is that it allows a company to leverage its existing investment in technology and personnel training. JDBC allows programmers to use their SQL skills to manipulate the data in the company's databases. As a matter of fact, JDBC allows direct access to a database server or access via database middleware. Furthermore, JDBC provides a way to connect to databases through an ODBC driver.

One advantage of JDBC over other middleware is that it requires no configuration on the client side. The JDBC driver is automatically downloaded and installed as part of the Java applet download. Because Java is a web-based technology, applications can connect to a database directly using a simple URL. Once the URL is invoked, the Java architecture comes into play, the necessary applets are downloaded to the client (including the JDBC database driver and all configuration information), and then the applets are executed securely in the client's runtime environment.

Every day, more and more companies are investing resources to develop and expand their web presence and are finding ways to do more business on the Internet. Such business generates increasing amounts of data to be stored in databases. Java and the .NET framework are part of the trend toward increasing reliance on the Internet as a critical business resource. In fact, the Internet has become a major development platform for most businesses. In the next section, you will learn more about Internet databases and how they are used.

Read

Database Systems: Design, Implementation, and Management, 13th Edition Carlos Coronel and Steven Morris, Chapter Fifteen page 693-704.

7.2 Database Internet Connectivity

Millions of people all over the world access the Internet and connect to databases via web browsers or data services. Internet database connectivity opens the door to new, innovative services that do the following:

- Permit rapid responses to competitive pressures.
- Increase customer satisfaction.
- Allow anywhere, anytime data.
- Yield fast and effective information dissemination.

Given these advantages, many organizations rely on their IT departments to create universal data access architectures based on Internet standards.

Database application development—particularly the creation and management of user interfaces and database connectivity—is profoundly affected by the web. The simplicity of the web's interface and its cross-platform functionality are at the core of its success as a data access platform. In fact, the web has helped create a new information dissemination standard. The following sections examine how web-to-database middleware enables end users to interact with databases over the web.

Web-to-Database Middleware: Server-Side Extensions

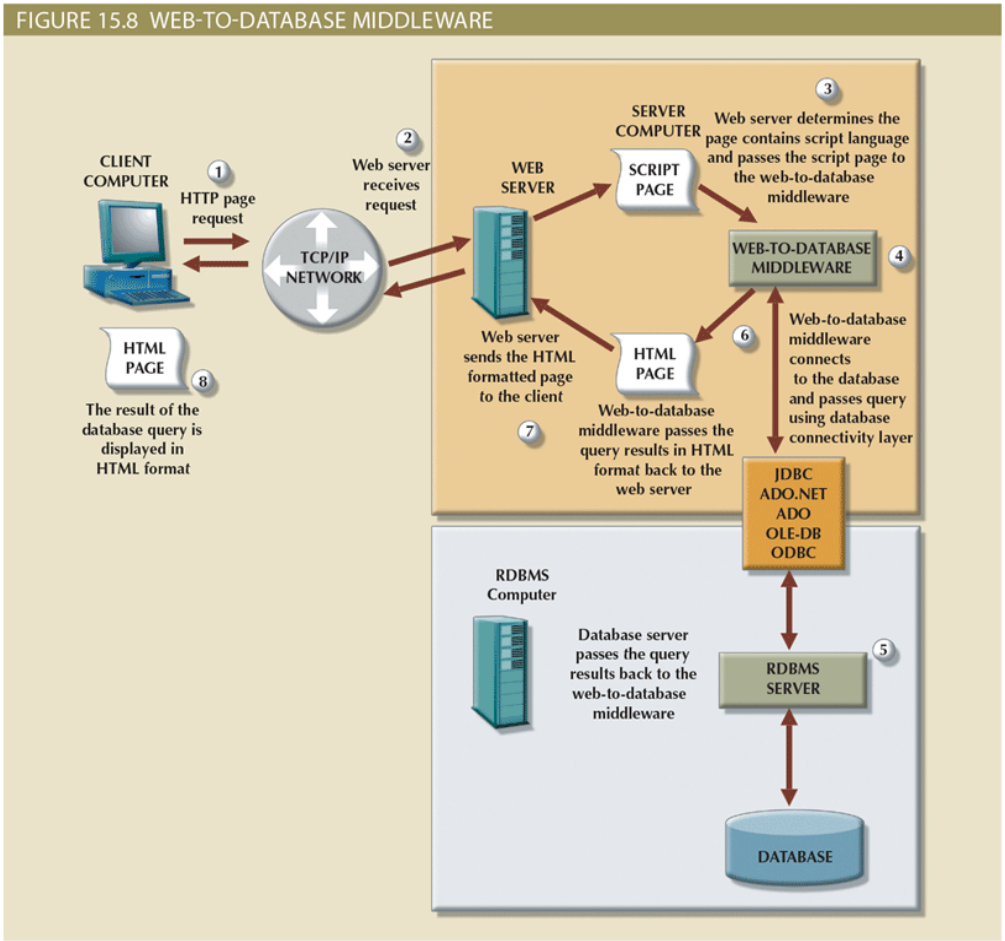
In general, the web server is the main hub through which all Internet services are accessed. For example, when an end user uses a web browser to dynamically query a database, the client browser requests a webpage from the web server. When the web server receives the page request, it looks for the page on the hard disk; when it finds the page, the server sends it back to the client.

Note that the web server generates the webpage contents before it sends the page to the client web browser. Ideally, the web server must include the database query result on the page before it sends that page back to the client. Unfortunately, neither the web browser nor the web server knows how to connect to and read data from the database. Therefore, to support this type of request, the web server's capability must be extended so it can understand and process database requests. This job is known as a server-side extension.

A **server-side extension** is a program that interacts directly with the web server to handle specific types of requests. The server-side extension makes it possible to retrieve and present the query results, but more importantly, it provides its services to the web server in a way that is totally transparent to the client browser. In short, the server-side extension adds significant functionality to the web server, and therefore to the Internet.

A database server-side extension program is also known as web-to-database middleware. Figure 15.8 shows the interaction between the browser, the web server, and the web-to-database middleware and from it you can trace the web-to-database middleware actions.

FIGURE 15.8 WEB-TO-DATABASE MIDDLEWARE



Web Server Interfaces

Extending web server functionality implies that the web server and the web-to-database middleware will properly communicate with each other. A web server interface defines a standard way to exchange messages with external programs. Currently, there are two well-defined web server interfaces:

- Common Gateway Interface (CGI)
- Application programming interface (API)

The **Common Gateway Interface** (CGI) uses script files that perform specific functions based on the client's parameters that are passed to the web server. The script file is a small program containing commands written in a programming language—usually Perl, C#, or Visual Basic. The script file's contents can be used to connect to the database and to retrieve data from it, using the parameters passed by the web server. Next, the script converts the retrieved data to HTML format and passes the data to the web server, which sends the HTML-formatted page to the client.

The main disadvantage of using CGI scripts is that the script file is an external program that executes separately for each user request and therefore causes a resource bottleneck. Performance also could be degraded by using an interpreted language or by writing the script inefficiently

An **application programming interface** (API) is a newer web server interface standard that is more efficient and faster than a CGI script. APIs are more efficient because they are implemented as shared code or as dynamic-link libraries (DLLs). That means the API is treated as part of the web server program that is dynamically invoked when needed.

APIs are faster than CGI scripts because the code resides in memory, so there is no need to run an external program for each request. Instead, the same API serves all requests. Another advantage is that an API can use a shared connection to the database instead of creating a new one every time, as is the case with CGI scripts.

Although APIs are more efficient in handling requests, they have some disadvantages. Because the APIs share the same memory space as the web server, an API error can bring down the web server. Another disadvantage is that APIs are specific to the web server and to the operating system.

Regardless of the type of web server interface used, the web-to-database middle-ware program must be able to connect with the database. That connection can be accomplished in one of two ways:

- Use the native SQL access middleware provided by the vendor.
- Use the services of general database connectivity standards.

The Web Browser

The web browser is software such as Microsoft Internet Explorer, Microsoft Edge, Google Chrome, Apple Safari, or Mozilla Firefox that lets end users navigate the web from their client computer. Each time the end user clicks a hyperlink, the browser generates an HTTP GET page request that is sent to the designated web server using the TCP/IP Internet protocol.

The web browser's job is to interpret the HTML code that it receives from the web server and to present the various page components in a standard formatted way. The web is a stateless system—at any given time, a web server does not know the status of any of the clients communicating with it. That is, there is no open communication line between the server and each client accessing it, which of course is impractical in a world wide web! Instead, client and server computers interact in very short "conversations" that follow the request-reply model. The only time the client and server computers communicate is when the client requests a page—when the user clicks a link—and the server sends the requested page to the client. Once the client receives the page and its components, the client/server communication is ended.

The web browser, through its use of HTML, does not have computational abilities beyond formatting output text and accepting form field inputs. Even when the browser accepts form field data, there is no way to perform immediate data entry validation. Therefore, to perform such crucial processing in the client, the web defers to other web programming languages such as Java, JavaScript, and VBScript. The browser resembles a dumb terminal that displays only data and can perform only rudimentary processing such as accepting form data inputs. To improve the capabilities of the web browser, you must use plug-ins and other client-side extensions. On the server side, web application servers provide the necessary processing power.

Client-Side Extensions

Client-side extensions add functionality to the web browser. Although client-side extensions are available in various forms, the most common are:

- Plug-ins
- Java and JavaScript
- ActiveX and VBScript

A **plug-in** is an external application that is automatically invoked by the browser when needed. The plug-in is associated with a data object—generally using the file extension—to allow the web server to properly handle data that is not originally supported.

From the developer's point of view, using routines that permit data validation on the client side is an absolute necessity. Therefore, client-side data input validation is one of the most basic requirements for web applications. Most of the data validation routines are done in Java, JavaScript, ActiveX, or VBScript.

Web Application Servers

A web application server is a middleware application that expands the functionality of web servers by linking them to a wide range of services, such as databases, directory systems, and search engines. The web application server also provides a consistent run-time environment for web applications. Web application servers can be used to perform the following:

- Connect to and query a database from a webpage.
- Present database data in a webpage using various formats.
- Create dynamic web search pages.
- Create webpages to insert, update, and delete database data.
- Enforce referential integrity in the application program logic.
- Use simple and nested queries and programming logic to represent business rules.

Web application servers provide features such as:

- An integrated development environment with session management and support for persistent application variables
- Security and authentication of users through user IDs and passwords
- Computational languages to represent and store business logic in the application server
- Automatic generation of HTML pages integrated with Java, JavaScript, VBScript, ASP, and so on
- Performance and fault-tolerant features
- Database access with transaction management capabilities
- Access to multiple services, such as file transfers (FTP), database connectivity, email, and directory services

Read

Database Systems: Design, Implementation, and Management, 13th Edition Carlos Coronel and Steven Morris, Chapter Fifteen page 704-715

7.3 Cloud Computing Services

You have almost certainly heard about the "cloud" from the thousands of publications and TV ads that have used the term over the years, although it has represented different concepts.

But what exactly is cloud computing? According to the National Institute of Standards and Technology (NIST), **cloud computing** is "a computing model for enabling ubiquitous, convenient, on-demand network access to a shared pool of configurable computer resources (e.g., networks, servers, storage, applications and services) that can be rapidly provisioned and released with minimal management effort or service provider interaction." The term cloud services is used in this book to refer to the services provided by cloud computing. Cloud services allow any organization to quickly and economically add information technology services such as applications, storage, servers, processing power, databases, and infra-structure to its IT portfolio. Figure 15.21 shows a representation of cloud computing services on the Internet.

Cloud computing eliminates financial and technological barriers so organizations can leverage database technologies in their business processes with minimal effort and cost. In fact, cloud services have the potential to turn basic IT services into "commodity" services such as electricity, gas, and water, and to enable a revolution that could change not only the way that companies do business, but the IT business itself.

Although some cloud services require some degree of customization on the customer's part, other cloud computing services are more transparent to the user and require less customization. For example, Dropbox and Microsoft OneDrive are simple cloud services that let you synchronize your documents, photos, music, and other files transparently over the Internet across many devices. As you can see, cloud computing implementations vary; the next section explains the basic types.

Cloud Implementation Types

Cloud computing has different types of implementations based on who the target customers are:

- **Public cloud**. This type of cloud infrastructure is built by a third-party organization to sell cloud services to the general public and is the most com-mon type of cloud implementation.
- **Private cloud**. This type of internal cloud is built by an organization for the sole purpose of servicing its own needs.
- **Community cloud**. This type of cloud is built by and for a specific group of organizations that share a common trade, such as agencies of the federal government, the military, or higher education.

Regardless of the implementation an organization uses, most cloud services share a common set of core characteristics. These characteristics are explored in the next section.

Characteristics of Cloud Services

Cloud computing services share a set of guiding principles. The prevalent characteristics are:

- Ubiquitous access via Internet technologies.
- Shared infrastructure. The cloud service infrastructure is shared by multiple users.
- Lower start-up costs and variable pricing.
- Flexible and scalable services.

- Dynamic provisioning. The consumer can quickly provision any needed resources.
- Service orientation. Cloud computing focuses on providing consumers with specific, well-defined services that use well-known interfaces.
- Managed operations.

The preceding list is not exhaustive, but it is a starting point to understand most cloud computing offerings. Although most companies move to cloud services because of cost savings, some companies move to them because they are the best way to gain access to specific IT resources that would otherwise be unavailable. Not all cloud services are the same; in fact, there are several different types, as explained in the next section.

Types of Cloud Services

Cloud services come in different shapes and forms; no single type of service works for all consumers. In fact, cloud services often follow an à la carte model; consumers can choose multiple service options according to their individual needs. Based on the types of services provided, cloud services can be classified by the following categories:

- *Software as a Service (SaaS)*. The cloud service provider offers turnkey applications that run in the cloud.
- *Platform as a Service (PaaS)*. The cloud service provider offers the capability to build and deploy consumer-created applications using the provider's cloud infrastructure.
- *Infrastructure as a Service (IaaS)*. In this case, the cloud service provider offers consumers the ability to provision their own resources on demand; these resources include storage, servers, databases, processing units, and even a complete virtualized desktop.

Cloud computing services have evolved in their sophistication and flexibility. The merging of new technologies has enabled the creation of new options such as "desktop as a service," which effectively creates a virtual computer on the cloud that can be accessed from any device over the Internet.

Cloud Services: Advantages and Disadvantages

Cloud computing has grown remarkably in the past few years. Companies of all sizes are enjoying the advantages of cloud computing, but its widespread adoption is still limited by several factors. Table 15.4 summarizes the main advantages and disadvantages of cloud computing.

TABLE 15.4

ADVANTAGES AND DISADVANTAGES OF CLOUD COMPUTING

ADVANTAGE	DISADVANTAGE
Low initial cost of entry. Cloud computing has lower costs of entry when compared with the alternative of building in house.	*Issues of security, privacy, and compliance*. Trusting sensitive company data to external entities is difficult for most data-cautious organizations.
Scalability/elasticity. It is easy to add and remove resources on demand.	*Hidden costs of implementation and operation*. It is hard to estimate bandwidth and data migration costs.
Support for mobile computing. Cloud computing providers support multiple types of mobile computing devices.	*Data migration is a difficult and lengthy process*. Migrating large amounts of data to and from the cloud infrastructure can be difficult and time-consuming.
Ubiquitous access. Consumers can access the cloud resources from anywhere at any time, as long as they have Internet access.	*Complex licensing schemes*. Organizations that implement cloud services are faced with complex licensing schemes and complicated service-level agreements.
High reliability and performance. Cloud providers build solid infrastructures that otherwise are difficult for the average organization to leverage.	*Loss of ownership and control*. Companies that use cloud services are no longer in complete control of their data. What is the responsibility of the cloud provider if data are breached? Can the vendor use your data without your consent?
Fast provisioning. Resources can be provisioned on demand in a matter of minutes with minimal effort.	*Organization culture*. End users tend to be resistant to change. Do the savings justify being dependent on a single provider? Will the cloud provider be around in 10 years?
Managed infrastructure. Most cloud implementations are managed by dedicated internal or external staff. This allows the organization's IT staff to focus on other areas.	*Difficult integration with internal IT system*. Configuring the cloud services to integrate transparently with internal authentication and other internal services could be a daunting task.

SQL Data Services

As you have seen in this chapter, data access technologies have evolved from simple ODBC data retrieval to advanced remote data processing using ADO.NET and XML. At the same time, companies are looking for ways to better manage ever-growing amounts of data while controlling costs without sacrificing data management features. Cloud computing provides a relatively stable and reliable platform for developing and deploying business services; cloud vendors have expanded their business to offer SQL data services. **SQL data services (SDS)** refers to a cloud computing-based data management service that provides relational data storage, access, and management to companies of all sizes without the typically high costs of in-house hardware, software, infrastructure, and personnel. This type of service provides some unique benefits:

- Hosted data management.
- Standard protocols.
- A common programming interface.

SQL data services offer the following advantages when compared with in-house systems:

- Highly reliable and scalable relational database for a fraction of the cost
- High level of failure tolerance because data is normally distributed and replicated among multiple servers
- Dynamic and automatic load balancing
- Automated data backup and disaster recovery included with the service
- Dynamic creation and allocation of database processes and storage

The use of SQL data services enables rapid application development for businesses with limited information technology resources, and allows them to rapidly deploy business solutions. A consumer of cloud services is free to use the database to create the best solution for the problem at hand. However, having access to relational database technology via a SQL data service is just the start — you still need to be knowledgeable in database design and SQL to develop high-quality applications.

Read

Database Systems: Design, Implementation, and Management, 13th Edition Carlos Coronel and Steven Morris, Chapter Fifteen, page 722-730

Chapter Summary/Review

In this chapter, you have learned:

- Database connectivity refers to the mechanisms through which application programs connect and communicate with data repositories. Database connectivity software is also known as database middleware.
- Microsoft database connectivity interfaces are dominant players in the market and enjoy the support of most database vendors. ODBC, OLE-DB, and ADO.NET form the backbone of Microsoft's Universal Data Access (UDA) architecture.
- Native database connectivity refers to the connection interface that is provided by the database vendor and is unique to that vendor.
- Object Linking and Embedding for Database (OLE-DB) is database middleware developed with the goal of adding object-oriented functionality for access to relational and non-relational data.
- Database access through the web is achieved through middleware. To improve the capabilities on the client side of the web browser, you must use plug-ins and other client-side extensions such as Java and JavaScript, or ActiveX and VBScript.
- Extensible Mark-up Language (XML) facilitates the exchange of B2B and other data over the Internet. It provides the semantics that facilitate the exchange, sharing, and manipulation of structured documents across organizational boundaries.
- Cloud computing is a computing model that provides ubiquitous, on-demand access to a shared pool of configurable resources that can be rapidly provisioned.
- SQL data services (SDS) refers to a cloud computing-based data management service that provides relational data storage, ubiquitous access, and local management to companies of all sizes.

Review Questions

1. What is the difference between DAO and RDO?
2. Explain the OLE-DB model based on its two types of objects.
3. What are web server interfaces used for? Give some examples.
4. What is a web application server, and how does it work from a database perspective?
5. What is XML, and why is it important?
6. Define SQL data services and list their advantages.

Problems

7. In the following exercises, you will set up database connectivity using MS Excel.
    a. Use MS Excel to connect to the Ch02_InsureCo MS Access database using ODBC, and retrieve all of the CUSTOMERs.
    b. Use MS Excel to connect to the Ch02_InsureCo MS Access database using ODBC, and retrieve the customers whose AGENT_CODE is equal to 503.
    c. Create a System DSN ODBC connection called Ch02_Tinycollege using the Administrative Tools section of the Windows Control Panel.
    d. Use MS Excel to list all classes taught in room KLR200 using the Ch02_TinyCollege System DSN.

To answer Problems 8−11, use Section 7-3a as your guide.

8. Create a sample XML document and DTD for the exchange of customer data.

9. Create a sample XML document and DTD for the exchange of product and pricing data.

10. Create a sample XML document and DTD for the exchange of order data.

11. Create a sample XML document and DTD for the exchange of student transcript data. Use your college transcript as a sample.

Review Questions (MCQ)

1. What does XML stand for?

    a. eXtra Modern Link
    b. eXtensible Markup Language
    c. Example Markup Language
    d. X-Markup Language

2. What is the correct syntax of the declaration which defines the XML version?

    a. <xml version="A.0" />
    b. <?xml version="A.0"?>
    c. <?xml version="A.0" />
    d. None of the above

3. Cloud computing has different types of implementations based on the target customers, which are:

    a. Public cloud, company cloud and private cloud
    b. Private cloud, community cloud and organization cloud
    c. Community cloud, private cloud and public cloud
    d. None of the above

4. Which of these is not a characteristic of cloud services?

    a. Ubiquitous access via Internet technologies.
    b. Higher start-up costs and fixed pricing.
    c. Dynamic provisioning.
    d. Flexible and scalable services.

5. Database connectivity software is also known as _____.

    a. Database connectors
    b. Firewalls
    c. DB routers
    d. Database middleware

6. The type of cloud service that provides the consumer with the capability to build deploy using the cloud service provider's infrastructure is called:

    a. Server as a Service (SaaS)
    b. Platform as a Service (PaaS)
    c. Infrastructure as a Service (IaaS)
    d. Software as a Service (SaaS)

LEARNING OUTCOMES

After reading this Section of the guide, the learner should be able to:

- Describe the impact of data quality on a company's assets and competitive position
- Describe the role of the database in supporting operational, tactical, and strategic decision-making
- Describe the impact that the introduction of a DBMS has on technological, managerial, and cultural aspects of an organization
- Describe the managerial and technical roles of the database administrator
- Describe the processes and systems in the information security framework that support the three database security goals
- Identify the standards, strategies, and tools used in database administration
- Describe the impact that cloud-based data services have on the role of the DBA

8.1 Data as an Asset and the Need for a Database

Data as a Corporate Asset

In Chapter 1, you learned that data is the raw material from which information is produced. Therefore, in today's information-driven environment, data is an asset that needs careful management.

To assess data's monetary value, consider what is stored in a company database: data about customers, suppliers, inventory, operations, and so on. How many opportunities are lost if the data is lost? What is the actual cost of data loss? Data loss puts any company in a difficult position.

Data is a valuable resource that can translate into information. If the information is accurate and timely, it can enhance the company's competitive position and generate wealth. In effect, an organization is subject to a data-information-decision cycle; that is, the data user applies intelligence to data to produce information that is the basis of knowledge used in decision making.

Most organizations continually seek new ways to leverage their data resources to get greater returns. This leverage can take many forms, from data warehouses to tighter integration with customers and suppliers in support of the electronic supply chain. As organizations become more dependent on information, that information's accuracy becomes more critical. **Dirty data**, or inaccurate and inconsistent data, becomes an even greater threat. Data can become dirty for many reasons:

- Lack of enforcement of integrity constraints, such as not null, uniqueness, and referential integrity
- Data-entry errors and typographical errors
- Use of synonyms and homonyms across systems
- Nonstandard use of abbreviations in character data
- Different decompositions of composite attributes into simple attributes across systems

Some causes of dirty data, such as improper implementation of constraints, can be addressed within an individual database. However, addressing other causes is more complicated. Some dirty data comes from the movement of data across systems, as in the creation of a data warehouse. Efforts to control dirty data are generally referred to as data quality initiatives.

Data quality is a comprehensive approach to ensuring the accuracy, validity, and timeliness of data. This comprehensive approach is important because data quality involves more than just cleaning dirty data; it also focuses on preventing future inaccuracies and building user confidence in the data. While data quality efforts vary greatly from one organization to another, most involve the following:

- A data governance structure that is responsible for data quality
- Measurements of current data quality
- Definition of data quality standards in alignment with business goals
- Implementation of tools and processes to ensure future data quality\

Several tools can assist in data quality initiatives. Data-profiling and master data management software are available from many vendors. **Data-profiling software** gathers statistics, analyses existing data sources and metadata to determine data patterns, and compares the patterns against standards that the organization has defined. **Master data management (MDM)** helps to prevent dirty data by coordinating common data across multiple systems. MDM software provides a "master" copy of entities that appear in numerous systems throughout the organization. While these

technological approaches provide an important part of data quality, the overall solution to high-quality data within an organization still relies heavily on data administration and management.

Read

Database Systems: Design, Implementation, and Management, 13th Edition Carlos Coronel and Steven Morris, Chapter Sixteen, page 735-736.

The Need for a Database and Its Role in an Organization

Data is used by different people in different departments for various reasons. Therefore, data management must address the concept of shared data.

Regardless of the organization, the database's predominant role is to *support managerial decision making at all levels in the organization while preserving data privacy and security*.

An organization's managerial structure might be divided into three levels: top-level management makes strategic decisions, middle management makes tactical decisions, and operational management makes daily working decisions.

The DBMS must give each level of management a useful view of the data and support the required level of decision making. The following activities are typical of each management level.

At the *top management* level, the database must be able to:

- Provide the information necessary for strategic decision making, strategic planning, policy formulation, and goals definition.
- Provide access to external and internal data to identify growth opportunities and to chart the direction of such growth.
- Provide feedback to monitor whether the company is achieving its goals.

At the *middle management* level, the database must be able to:

- Deliver the data necessary for tactical decisions and planning.
- Monitor and control the allocation and use of company resources and evaluate the performance of various departments.
- Provide a framework for enforcing and ensuring the security and privacy of the data in the database.

At the *operational management* level, the database must be able to:

- Represent and support company operations as closely as possible. Produce query results within specified performance levels.
- Enhance the company's short-term operations by providing timely information for customer support and for application development and computer operations.

A general objective for any database is to provide a seamless flow of information throughout the company.

The company's database is also known as the corporate or enterprise database. The enterprise database might be defined as the company's data representation that provides support for all present and expected future operations—from design to implementation, from sales to services, and from daily decision making to strategic planning.

Read

Database Systems: Design, Implementation, and Management, 13th Edition Carlos Coronel and Steven Morris. Chapter Sixteen, page 736-738.

Introduction of a Database: Special Considerations

Having a computerized database management system does not guarantee that the data will be properly used to provide the best solutions required by managers. A DBMS is a tool for managing data; like any tool, it must be used effectively to produce the desired results.

The introduction of a DBMS represents a big change and challenge, which is likely to have a profound impact on an organization, which might be positive or negative depending on how it is administered.

The introduction of a DBMS has been described as a process that includes three important aspects:

- *Technological*—DBMS software and hardware
- *Managerial*—Administrative functions
- *Cultural*—Corporate resistance to change

The *technological* aspect includes selecting, installing, configuring, and monitoring the DBMS as well as employing competent technically skilled administrative staff to make sure that it efficiently handles data storage, access, and security.

With regards to the *managerial* aspect, it must be understood that a high-quality DBMS does not guarantee a high-quality information system.

The introduction of a DBMS requires careful planning to create an appropriate organizational structure and accommodate the personnel responsible for administering the system. This structure must also be subject to well-developed monitoring and controls. The administrative personnel must have excellent interpersonal and communications skills combined with broad organizational and business understanding.

The DBMS is likely to have an effect on people, functions, and interactions. A *cultural* impact is likely because the database approach creates a more controlled and structured information flow.

When the new database comes online, people might be reluctant to use its information and might question its value or accuracy. Many might be disappointed that the information does not fit their

preconceived notions and strongly held beliefs. Database administrators must be prepared to open their doors to end users, listen to their concerns, act on those concerns when possible, and explain the system's uses and benefits.

Read

Database Systems: Design, Implementation, and Management, 13th Edition Carlos Coronel and Steven Morris, Chapter Sixteen, page 738-739.

8.2 The Evolution of Database Administration and the Database Administrator

The Evolution of Database Administration

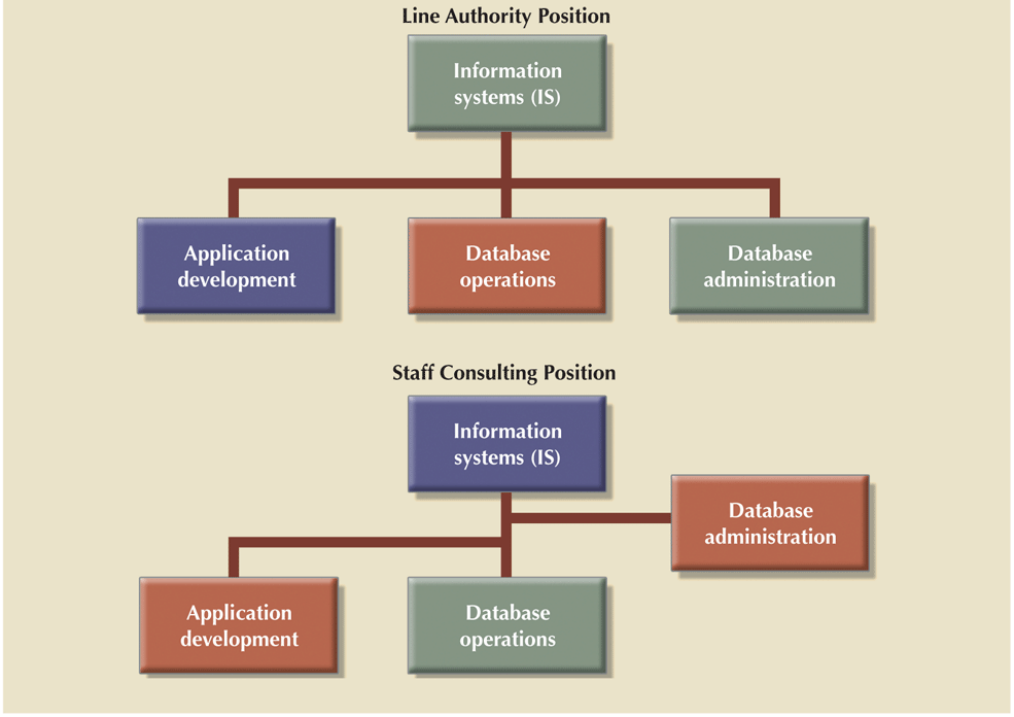
Data administration has its roots in the old, decentralized world of the file system. The cost of data and managerial duplication in these systems gave rise to centralized data administration known as the electronic data processing (EDP) or data processing (DP) department. DP administrators were given the authority to manage all company file systems as well as resolve data and managerial conflicts created by the duplication and misuse of data. The advent of the DBMS and its shared view of data produced a new level of data management sophistication and led the DP department to evolve into an **information systems (IS) department**. The responsibilities of the IS department were broadened to include the following:

- A *service* function to provide end users with data management support
- A *production* function to provide end users with solutions for their information needs through integrated application or management information systems

As demand grew, the IS application development segment was subdivided by the type of system it supported: accounting, inventory, marketing, data warehousing, business intelligence, and so on. However, this development meant that database administration responsibilities were divided. The application development segment was in charge of gathering database requirements and logical database design, whereas the database operations segment took charge of implementing, monitoring, and controlling DBMS operations. As the number of database applications grew, data management became increasingly complex, thus leading to the development of database administration. The person responsible for control of the centralized and shared database became known as the **database administrator (DBA)**.

The size and role of the DBA function varies from company to company, as does its placement within the organizational structure. On the organizational chart, the DBA function might be defined as either a staff or line position. The two possible DBA positions are illustrated in Figure 16.3.

FIGURE 16.3 THE PLACEMENT OF THE DBA FUNCTION



There is no standard for how the DBA function fits in an organization's structure, partly because the function itself is probably the most dynamic of any in an organization. In fact, the fast-paced changes in DBMS technology dictate changing organizational styles. For example:

- The development of distributed databases can force an organization to decentralize data administration further, in turn imposing new and more complex coordinating activities on the system DBA.
- The increasing use of cloud data services is pushing many database platforms and infrastructures into the cloud. In such environments, the DBA becomes a data use service provider and advisor for the organization.
- Conversely, the growing use of Big Data in organizations can force the DBA to become more technology-oriented.

DBA operations are commonly defined and divided according to the phases of the Database Life Cycle (DBLC). If that approach is used, the DBA function requires personnel to cover the following activities, and more:

- Database planning, including the definition of standards, procedures, and enforcement
- Database requirements gathering and conceptual design
- Database physical design and implementation
- Database testing and debugging
- Data quality monitoring and management

Keep in mind that a company might have several incompatible DBMSs installed to systems administrator support different operations. A variety of personal computer DBMSs might be installed in different departments. In such an environment, the company might have one DBA assigned for each DBMS. The general coordinator of all DBAs is sometimes known as the **systems administrator**.

There is a growing trend toward specialization in data management. For example, the organizational charts used by some larger corporations make a distinction between a DBA and the **data administrator (DA)**. The DA, also known as the **information resource manager (IRM)**, usually reports directly to top management and is given a higher degree of responsibility and authority than the DBA, although the two roles can overlap.

Read

Database Systems: Design, Implementation, and Management, 13th Edition Carlos Coronel and Steven Morris, Chapter Sixteen, page 739-743.

The Database Environment's Human Component

The most carefully crafted database system cannot operate without human assistance. In this section, you will explore how people perform the data administration activities that make a good database design useful.

Table 16.1 contrasts the general characteristics of both positions by summarizing typical DA and DBA activities. All of these activities are assigned to the DBA if the organization does not employ both a DA and a DBA.

TABLE 16.1

CONTRASTING DA AND DBA ACTIVITIES AND CHARACTERISTICS

DATA ADMINISTRATOR (DA)	DATABASE ADMINISTRATOR (DBA)
Performs strategic planning	Controls and supervises
Sets long-term goals	Executes plans to reach goals
Sets policies and standards	Enforces policies and procedures Enforces programming standards
Job is broad in scope	Job is narrow in scope
Focuses on the long term	Focuses on the short term (daily operations)
Has a managerial orientation	Has a technical orientation
Is DBMS-independent	Is DBMS-specific

The goals that the DA sets are defined by issues such as:

- Data "sharability" and time availability
- Data consistency and integrity
- Data security and privacy

Naturally, the list can be expanded to fit an organization's specific data needs. Regardless of how data management is conducted—and despite the fact that great authority is invested in the DA or DBA to define and control the way company data is used—the DA and DBA do not own the data.

The arbitration of interactions between the two most important assets of any organization, people and data, places the DBA in a dynamic environment where the DBA is the focal point for data and user interaction.

The DBA defines and enforces the procedures and standards to be used by programmers and end users during their work with the DBMS. The DBA also verifies that programmer and end-user access meets the required quality and security standards.

Database users might be classified by the following criteria:

- Type of decision-making support required (operational, tactical, or strategic)
- Degree of computer knowledge (novice, proficient, or expert)
- Frequency of access (casual, periodic, or frequent)

These classifications are not exclusive and usually overlap. The DBA must be able to interact with all of them, understand their different needs, answer questions at all levels of expertise, and communicate effectively. The DBA activities emphasises the need for a diverse mix of skills.

The skills of the DBA can be divided into two categories—managerial and technical—as summarized in Table 16.2.

TABLE 16.2

DESIRED DBA SKILLS

MANAGERIAL	TECHNICAL
Broad business understanding	Broad data-processing background and up-to-date knowledge of database technologies
Coordination skills	Understanding of Systems Development Life Cycle
Analytical skills	Structured methodologies • Data flow diagrams • Structure charts • Programming languages
Conflict resolution skills	Knowledge of Database Life Cycle
Communication skills (oral and written)	Database modeling and design skills • Conceptual • Logical • Physical
Negotiation skills	Operational skills: Database implementation, data dictionary management, security, and so on
Experience: 10 years in a large DP department	

As you examine Table 16.2, keep in mind that the DBA must perform two distinct roles. The DBA's managerial role is focused on personnel management and on interactions with end users. The DBA's technical role involves the use of the DBMS—database design, development, and implementation—as well as the production, development, and use of application programs.

More specifically, the DBA's responsibilities (divided into activities and services) are shown in Table 16.3.

TABLE 16.3

DBA ACTIVITIES AND SERVICES

DBA ACTIVITY		DBA SERVICE
Planning		End-user support
Organizing		Policies, procedures, and standards
Testing	of	Data security, privacy, and integrity
Monitoring		Data backup and recovery
Delivering		Data distribution and use

The following sections examine the services in greater detail.

End-User Support The DBA interacts with end users by providing data and information support services which include the following:

- Gathering user requirements. The DBA must work with end users to help gather the data required to identify and describe their present and future information needs.
- Building end-user confidence. Finding adequate solutions to end users' problems increases their trust and confidence in the DBA.
- Resolving conflicts and problems. When conflicts arise, the DBA must have the authority and responsibility to resolve them.
- Finding solutions to information needs. The ability and authority to resolve data conflicts enables the DBA to develop solutions that will properly fit within the data management framework and address end users' information needs.
- Ensuring quality and integrity of data and applications. The DBA must work with application programmers and end users to teach them the database standards and procedures required for data quality, access, and manipulation.
- Managing the training and support of DBMS users. The DBA must ensure that all users understand the basic functions of the DBMS software.

Policies, Procedures, and Standards A successful data administration strategy requires the continuous enforcement of policies, procedures, and standards for correct data creation, usage, and distribution within the database. The DBA must define, document, and communicate the following before they can be enforced:

- **Policies** are general statements of direction or action that communicate and support DBA goals.
- **Standards** describe the minimum requirements of a given DBA activity; they are more detailed and specific than policies.
- **Procedures** are written instructions that describe a series of steps to be followed during the performance of a given activity.

The defined procedures cover areas such as:

- End-user database requirements gathering.
- Database design and modelling.
- Documentation and naming conventions.
- Design, coding, and testing of database application programs.
- Database software selection.
- Database security and integrity.
- Database backup and recovery.
- Database maintenance and operation.
- End-user training.

Procedures and standards must be revised at least annually to keep them up to date and to ensure that the organization can adapt quickly to changes in the work environment.

Data Security, Privacy, and Integrity Data security, privacy, and integrity are of great concern to DBAs who manage DBMS installations. The DBA must use the security and integrity mechanisms provided by the DBMS to enforce the database administration policies defined in the previous section. In addition, DBAs must team up with Internet security experts to build security mechanisms that safeguard data from possible attacks or unauthorized access.

Data Backup and recovery When data is not readily available, companies face potentially ruinous losses. Therefore, data backup and recovery procedures are critical in all database installations. The DBA must also ensure that data can be fully recovered in case of data loss or loss of database integrity.

The management of database security, integrity, backup, and recovery is so critical that many DBA departments have created a position called the **database security officer (DSO)**. The DSO's sole job is to ensure database security and integrity. In large organizations, the DSO's activities are often classified as disaster management.

Backup and recovery measures must include at least the following:

- Periodic data and application backups.
- Proper backup identification.
- Convenient and safe backup storage
- Physical protection of both hardware and software.
- Personal access control to the software of a database installation.
- Insurance coverage for the data in the database.
- Data recovery and contingency plans must be thoroughly tested and evaluated, and they must be practiced frequently.
- A backup and recovery program is not likely to cover all components of an information system.

Note that for the DBA to effectively perform his/her administrative duties and provide these services, it requires technical and managerial skills, as well as people skills. In addition, note the overlap of the concepts involved in their managerial and technical roles. The managerial role, requires knowledge of the service areas mentioned in the sections above and the technical role involves application of that knowledge, where the DBA is hands on with design and implementation of the database system.

Read

Database Systems: Design, Implementation, and Management, 13th Edition Carlos Coronel and Steven Morris, Chapter Sixteen, page 743-757.

8.3 Security

Information system security refers to activities and measures that ensure the confidentiality, integrity, and availability of an information system and its main asset, data. Securing data requires a comprehensive, company-wide approach. To understand the scope of data security, consider each of the three security goals in more detail:

- **Confidentiality** deals with ensuring that data is protected against unauthorized access, and if the data is accessed by an authorized user, that it is used only for an authorized purpose. Data must be evaluated and classified according to the level of confidentiality: highly restricted (very few people have access), confidential (only certain groups have access), and unrestricted (can be accessed by all users). The data security officer spends a great amount of time ensuring that the organization is in compliance with desired levels of confidentiality. **Compliance** refers to activities that meet data privacy and security reporting guidelines.
- **Integrity**, within the data security framework, is concerned with keeping data consistent and free of errors or anomalies. The DBMS plays a pivotal role in ensuring the integrity of the data in the database. However, from the security point of view, the organizational processes, users, and usage patterns also must maintain integrity. Maintaining data integrity is a process that starts with data collection and continues with data storage, processing, usage, and archiving. The rationale behind integrity is to treat data as the most-valuable asset in the organization and to ensure that rigorous data validation is carried out at all levels within the organization.
- **Availability** refers to the accessibility of data whenever required by authorized users and for authorized purposes. To ensure data availability, the entire system must be protected from service degradation or interruption caused by any internal or external source.

Security Policies

Normally, the tasks of securing the system and its main asset, the data, are performed by the database security officer and the database administrator(s), who work together to establish a cohesive data security strategy. Such a strategy begins with defining a sound and comprehensive security policy. A **security policy** is a collection of standards, policies, and procedures created to guarantee the security of a system and ensure auditing and compliance. The security audit process starts by identifying security vulnerabilities in the organization's information system infrastructure and identifying measures to protect the system and data against those vulnerabilities.

Security Vulnerabilities

A **security vulnerability** is a weakness in a system component that could be exploited to allow unauthorized access or cause service disruptions. Such vulnerabilities could fall under one of the following categories:

- *Technical*. An example would be a flaw in the operating system or web browser.
- *Managerial*. For example, an organization might not educate users about critical security issues.
- *Cultural*. Users might hide passwords under their keyboards or forget to shred confidential reports.
- *Procedural*. Company procedures might not require complex passwords or the checking of user IDs.

When a security vulnerability is left unchecked, it could become a security threat. A **security threat** is an imminent security violation.

A **security breach** occurs when a security threat is exploited to endanger the integrity, confidentiality, or availability of the system. Security breaches can lead to a database whose integrity is either preserved or corrupted:

- *Preserved*. In these cases, action is required to avoid the recurrence of similar security problems, but data recovery may not be necessary.
- *Corrupted*. Action is required to avoid the recurrence of similar security problems, and the database must be recovered to a consistent state.

Database Security

Database security refers to DBMS features and other related measures that comply with the organization's security requirements. From the DBA's point of view, security measures should be implemented to protect the DBMS against service degradation and to protect the database against loss, corruption, or mishandling.

To protect the DBMS against service degradation, some security safeguards are recommended. For example:

- Change default system passwords.
- Change default installation paths.
- Set up auditing logs.
- Set up session logging.
- Require session encryption.

Furthermore, the DBA should work closely with the network administrator to implement network security that protects the DBMS and all services running on the network.

Protecting the data in the database is a function of authorization management. **Authorization management** defines procedures to protect and guarantee database security and integrity. Those procedures include the following:

- *User access management.* This function is designed to limit access to the database; it likely includes at least the following procedures:
    - Define each user to the database.
    - Assign passwords to each user.
    - Define user groups.
    - Assign access privileges.
    - Control physical access.
- *View definition*. The DBA must define data views to protect and control the scope of the data that are accessible to an authorized user.
- *DBMS access control.* Database access can be controlled by placing limits on the use of DBMS query and reporting tools.
- *DBMS usage monitoring*. The DBA must also audit the use of data in the database. Several DBMS packages contain features that allow the creation of an **audit log**, which automatically records a brief description of database operations performed by all users.

The integrity of a database could be lost because of external factors beyond the DBA's control. For example, the database might be damaged or destroyed by an explosion, a fire, or an earthquake.

Whatever the reason, the spectre of database corruption or destruction makes backup and recovery procedures crucial to any DBA.

Read

Database Systems: Design, Implementation, and Management, 13th Edition Carlos Coronel and Steven Morris, Chapter Sixteen, page 757-761.

8.4 Database Administration Tools

The extraordinary growth of data management activities within organizations created the need for better management standards, processes, and tools. Over the years, a new industry arose dedicated exclusively to data administration tools. These tools cover the entire spectrum of data administration tasks, from selection to inception, deployment, migration, and day-to-day operations. For example, you can find sophisticated data administration tools for:

- Database monitoring
- Database performance tuning
- SQL code optimization
- Database data extraction, transformation, and loading

The list above is not an exhaustive list of tools but the tools have something in common. They all expand the database's metadata or data dictionary. This section examines the data dictionary as a data administration tool, as well as the DBA's use of computer-aided systems engineering (CASE) tools to support database analysis and design.

The Data Dictionary

The DBMS *data dictionary* provides the DBMS with its self-describing characteristic. In effect, the data dictionary resembles an x-ray of the company's entire data set, and it is a crucial element in data administration.

Two main types of data dictionaries exist: *integrated* and *standalone*. An integrated (built-in) data dictionary is included with the DBMS. Other DBMSs, especially older types, do not have a built-in data dictionary; instead, the DBA may use third-party *standalone* systems.

Data dictionaries can also be classified as *active* or *passive*. An **active data dictionary** is automatically updated by the DBMS with every database access to keep its access information up to date. A **passive data dictionary** is not updated automatically and usually requires running a batch process. Data dictionary access information is normally used by the DBMS for query optimization.

The data dictionary's main function is to store the description of all objects that interact with the database. Integrated data dictionaries tend to limit their metadata to the data managed by the DBMS. Standalone data dictionary systems are usually more flexible and allow the DBA to describe and manage all of the organization's data, whether they are computerized or not. Whatever the data dictionary's format, it provides database designers and end users with a much-improved ability to communicate. In addition, the data dictionary is the tool that helps the DBA resolve data conflicts.

Although there is no standard format for the information stored in the data dictionary, several features are common. For example, the data dictionary typically stores descriptions of the following:

- Data elements that are defined in all tables of all databases
- Tables defined in all databases.
- Indexes defined for each database table.
- Defined databases.
- End users and administrators of the database.
- Programs that access the database.
- Access authorizations for all users of all databases.
- Relationships among data elements.

If the data dictionary can be organized to include data external to the DBMS itself, it becomes an especially flexible tool for more general corporate resource management and can then be described as the **information resource dictionary**.

The metadata stored in the data dictionary is often the basis for monitoring database use and for assigning access rights to database users. The information stored in the data dictionary is usually based on a relational table format, thus enabling the DBA to query the database with SQL commands. In the following section, the IBM DB2 system catalogue tables are the basis for several examples of how a data dictionary is used to derive information:

- SYSTABLES stores one row for each table or view.
- SYSCOLUMNS stores one row for each column of each table or view.
- SYSTABAUTH stores one row for each authorization given to a user for a table or view in a database.

Examples of Data Dictionary Usage

Example 1

List the names and creation dates of all tables created by the user JONESVI in the current database

```
SELECT NAME, CTIME

FROM SYSTABLES

WHERE CREATOR = 'JONESVI';
```

Example 2

List the names of the columns for all tables created by JONESVI in the current database.

```
SELECT NAME

FROM SYSCOLUMNS

WHERE TBCREATOR = 'JONESVI';
```

Example 3

List the names of all users who have some type of authority over the INVENTORY table.

```
SELECT DISTINCT GRANTEE
FROM SYSTABAUTH
WHERE TTNAME = 'INVENTORY';
```

As you can see in the preceding examples, the data dictionary can be a tool for monitoring database security by checking the assignment of data access privileges.

The data dictionary can also be used to ensure that application programmers have met the naming standards for data elements in the database, and that the data validation rules are correct. Thus, the data dictionary can be used to support a wide range of data administration activities and facilitate the design and implementation of information systems. Integrated data dictionaries are also essential to the use of computer-aided systems engineering tools.

Case Tools

A **computer-aided systems engineering** (CASE) tool provides an automated framework for the Systems Development Life Cycle (SDLC). **CASE** uses structured methodologies and powerful graphical interfaces. Because they automate many tedious system design and implementation activities, CASE tools play an increasingly important role in information systems development.

CASE tools are usually classified according to the extent of support they provide for the SDLC. The benefits associated with CASE tools include:

- A reduction in development time and costs
- Automation of the SDLC
- Standardization of systems development methodologies
- Easier maintenance of application systems developed with CASE tools

One of the CASE tools' most important components is an extensive data dictionary, which keeps track of all objects created by the systems designer. A CASE data dictionary also describes the relationships among system components.

Several CASE tools provide interfaces that work with the DBMS and allow the CASE tool to store its data dictionary information using the DBMS. Such interaction demonstrates the interdependence that exists between systems development and database development, and it helps create a fully integrated development environment.

In a CASE development environment, database and application designers use the CASE tool to store the description of the database schema, data elements, application processes, screens, reports, and other data relevant to development.

As an additional benefit, a CASE environment tends to improve the extent and quality of communication among the DBA, application designers, and end users. When the CASE tool finds conflicts, rules violations, and inconsistencies, it facilitates making corrections. Better yet, the CASE tool can make a correction and then cascade its effects throughout the applications environment, which greatly simplifies the job of the DBA and the application designer.

A typical CASE tool provides five components:

- Graphics designed to produce structured diagrams
- Screen painters and report generators to produce the information system's input and output formats
- An integrated repository for storing and cross-referencing the system design data
- An analysis segment to provide a fully automated check on system consistency, syntax, and completeness
- A program documentation generator

Most CASE tools, produce fully documented ER diagrams that can be displayed at different abstraction levels.

Major relational DBMS vendors, such as Oracle, now provide fully integrated CASE tools for their own DBMS software as well as for RDBMSs supplied by other vendors. Some vendors even take non-relational DBMSs, develop their schemas, and produce the equivalent relational designs automatically. Table 16.5 shows a short list of the many available CASE Data Modelling tool vendors.

TABLE 16.5

CASE DATA MODELING TOOLS

COMPANY	PRODUCT	WEBSITE
Casewise	Corporate Modeler Suite	*www.casewise.com*
Erwin Inc.	ERwin Data Modeler	*www.erwin.com*
Idera Inc.	ER/Studio Data Architect	*www.embarcadero.com/products/er-studio-data-architect*
Microsoft	Visio	*office.microsoft.com/en-us/visio*
Oracle	SQL Developer Data Modeler	*www.oracle.com/technetwork/developer-tools/datamodeler/overview/index.html*
IBM	Rational Software Architect Designer	*www-01.ibm.com/software/rational/products/swarchitect/*
SAP	Power Designer	*www.sap.com/products/powerdesigner-data-modeling-tools.html*
Visible Systems	Visible Analyst	*www.visible.com/Products/Analyst*

There is no doubt that CASE tools have enhanced the efficiency of database designers and application programmers. However, no matter how sophisticated the CASE tool, its users must be well versed in conceptual design. In the hands of database novices, CASE tools produce impressive-looking but bad designs.

Read

Database Systems: Design, Implementation, and Management, 13th Edition Carlos Coronel and Steven Morris, Chapter Sixteen, page 761-767.

8.5 Developing a Database Administration Strategy

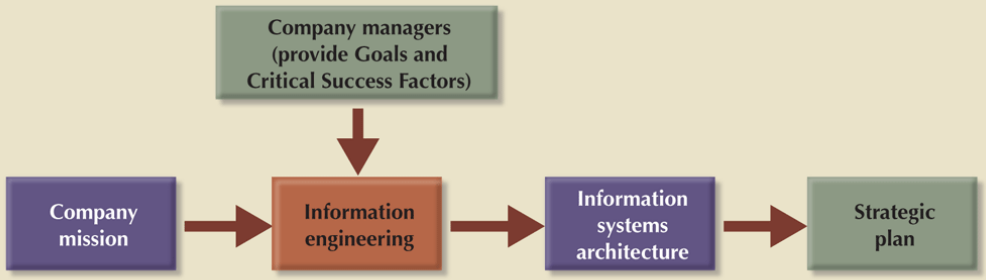
For a company to succeed, its activities must be committed to its main objectives or mission. Therefore, regardless of its size, a critical step for any organization is to ensure that its information system supports its strategic plans for each business area.

The database administration strategy must not conflict with the information systems plans. Several methodologies are available to ensure the compatibility of data administration and information systems plans and to guide strategic plan development. The most commonly used methodology is known as information engineering.

Information engineering (IE) allows for translation of the company's strategic goals into the data and applications that will help the company achieve those goals. IE focuses on the description of corporate data instead of the processes and by so doing, helps decrease the impact on systems when processes change.

The output of the IE process is an **information systems architecture (ISA)** that serves as the basis for planning, development, and control of future information systems. Figure 16.8 shows the forces that affect ISA development.

FIGURE 16.8 FORCES AFFECTING THE DEVELOPMENT OF THE ISA



Implementing IE in an organization involves planning, a commitment of resources, management liability, well-defined objectives, identification of critical factors, and control. An ISA provides a framework that includes computerized, automated, and integrated tools such as a DBMS and CASE tools.

The success of the overall information systems strategy and data administration strategy depends on several critical success factors that the DBA needs to understand. Critical success factors include the following managerial, technological, and corporate culture issues:

- Management commitment.
- Thorough analysis of the company situation.
- End-user involvement.
- Defined standards.
- Training.
- A small pilot project.

This list of factors is not comprehensive, but it does provide the framework for developing a successful strategy. Remember that no matter how comprehensive you make the list, it must be based on developing and implementing a data administration strategy that is tightly integrated with the organization's overall information systems planning.

Read

Database Systems: Design, Implementation, and Management, 13th Edition Carlos Coronel and Steven Morris, Chapter Sixteen, page 769-777.

Chapter Summary/Review

- Data management is a critical activity for any organization, so data must be treated as a corporate asset.
- Data quality is a comprehensive approach to ensure the accuracy, validity, and timeliness of data. Data quality focuses on correcting dirty data, preventing future inaccuracies in the data, and building user confidence in the data.
- The DBMS is the most commonly used tool for corporate data management. The DBMS supports strategic, tactical, and operational decision making at all levels of the organization.
- The database administrator (DBA) is responsible for managing the corporate database. Although no standard exists, it is common practice to divide DBA operations according to phases of the Database Life Cycle.
- The DA and DBA functions tend to overlap. Generally speaking, the DA has more managerial tasks than the more technically oriented DBA.
- A DBA's managerial services include supporting end users; defining and enforcing policies, procedures, and standards for the database; ensuring data security, privacy, and integrity; providing data backup and recovery services; and monitoring distribution and use of the data in the database.
- The DBA's technical role requires involvement in at least the following activities: evaluating, selecting, and installing the DBMS; designing and implementing databases and applications; testing and evaluating databases and applications; operating and maintaining the DBMS, utilities, and applications; and training and supporting users.
- Security refers to activities and measures that ensure the confidentiality, integrity, and availability of an information system and its main asset, data. A security policy is a collection of standards, policies, and practices that guarantee the security of a system and ensure auditing and compliance.
- A security vulnerability is a weakness in a system component that could be exploited to allow unauthorized access or service disruption. A security threat is an imminent security violation caused by an unchecked vulnerability. It is critical to have robust database security.
- The development of a data administration strategy is closely related to the company's mission and objectives. Therefore, the strategic plan requires a detailed analysis of company goals, its situation, and its business needs. The most commonly used integrating methodology, to guide the development of this data administration plan, is known as information engineering (IE).
- To help translate strategic plans into operational plans, the DBA has access to an arsenal of database administration tools, including a data dictionary and computer-aided systems engineering (CASE) tools.
- With the introduction of reliable cloud-based data services, the role of the DBA has expanded beyond corporate walls.

Review Questions

1. Define dirty data, and identify some of its sources.
2. Define security and privacy. How are the two concepts related?
3. Describe the DBA's responsibilities.
4. What DBA activities support end users?
5. What are the levels of data confidentiality?

Problems

6. Discuss the importance and characteristics of database backup and recovery procedures. Then describe the actions that must be detailed in backup and recovery plans.
7. Why are testing and evaluation of the database and applications not done by the same people who are responsible for design and implementation? What minimum standards must be met during testing and evaluation?
8. What are security vulnerabilities? What is a security threat? Give some examples of security vulnerabilities in different IS components.
9. Briefly explain the concepts of information engineering (IE) and information systems architecture (ISA). How do those concepts affect the data administration strategy?
10. Identify and explain some critical success factors in the development and implementation of a good data administration strategy.

Review Questions (MCQ)

1. Poor data administration can lead to which of the following?

   a) A single definition of the same data entity
   b) Poor decision making
   c) Missing data elements
   d) All of the above

2. Which of the following is not the data administrator's responsibility

   a) Establishing backup and recovery procedures
   b) Defining business rules
   c) Tuning database performance
   d) Defining and implementing database security policies and standards

3. Which of the following could result in data being unavailable?

   a) Data becoming lost or inaccurate
   b) The database server
   c) Planned database maintenance activities
   d) All of the above

4. Databases are not the property of a single function or individual within the organization.

   a) True
   b) False

5. Which of the following is part of an administrative policy to secure a database?

   a) Authentication policies
   b) Limiting particular areas within a building to only authorized people
   c) Ensure appropriate responses rates are in external maintenance agreements
   d) None of the above

6. Information repositories are replacing data dictionaries in many organizations.

   a) True
   b) False

References

- Foote, K. (December 21, 2016). *A Review of Different Database Types: Relational versus Non-Relational - DATAVERSITY*. [online] DATAVERSITY. Available at: https://www.dataversity.net/review-pros-cons-different-databases-relational-versus-non-relational/#.
- Ridwan, M. and Loy, Z. (1978). *A data abstraction model*. [online] Ir.canterbury.ac.nz. Available at: https://ir.canterbury.ac.nz/handle/10092/12979.
- Barry, D. (n.d.). *Relational Model Concepts*. [online] Service Architecture. Available at: https://www.service-architecture.com/articles/database/relational_model_concepts.html.
- Tahaghoghi, S. and Williams, H. (2007). *Learning MySQL*. Sebastopol, Calif.: O'Reilly.
- Ambler, S. (2004). *Introduction to Data Normalization: A Database "Best" Practice*. [online] Agiledata.org. Available at: http://agiledata.org/essays/dataNormalization.html.
- Lott, S. (2006). *Database Design: How Table Normalization Can Improve Performance*. [online] Dr. Dobb's. Available at: http://www.drdobbs.com/architecture-and-design/database-design-how-table-normalization/186701027.
- Hartmann, Sven & Link, Sebastian. (2003). *On Functional Dependencies in Advanced Data Models.* Electr. Notes Theor. Comput. Sci.. 84. 117-128. 10.1016/S1571-0661(04)80849-1.
- Al-Qaimari, Ghassan & W Paton, Norman & C Kilgour, Alistair. (1994). *Visualizing advanced data modelling constructs.* Information and Software Technology. 36. 597-605. 10.1016/0950-5849(94)90019-1.
- Tillmann, George. (2017). *Usage-Driven Database Design*. 10.1007/978-1-4842-2722-0.
- A. Nimalasena and V. Getov. (2014). *Performance tuning of database systems using a context-aware approach.* 9th International Conference on Computer Engineering & Systems (ICCES), Cairo, 2014, pp. 98-103.doi: 10.1109/ICCES.2014.7030936
- V. K. Myalapalli, A. Chakravarthy and K. P. Reddy. (2015). *Accelerating SQL queries by unravelling performance bottlenecks in DBMS engine*. International Conference on Energy Systems and Applications, Pune, 2015, pp. 7-12.doi: 10.1109/ICESA.2015.7503304
- L. Harrington, Jan. (2016). *Database Design and Performance Tuning*. 10.1016/B978-0-12-804399-8.00008-9.
- K. Munir. (2015). *Security model for cloud database as a service (DBaaS)*. International Conference on Cloud Technologies and Applications (CloudTech), Marrakech, 2015, pp. 1-5.doi: 10.1109/CloudTech.2015.7336974
- Jain, N., Raghu, D. and Kanna, D. (2018). *DaaS (Database as a Service) in Cloud Computing*. International Journal of Innovations & Advancement in Computer Science ,ISSN 2347-8616.[online] Academicscience.co.in. Available at: http://academicscience.co.in/admin/resources/project/paper/f201803221521727988.pdf.
- E. Walters, Robert & Fritchey, Grant & Taglienti, Carmen. (2009). *The Database Administration Profession*. 10.1007/978-1-4302-2414-3_1.
- J. C. Odirichukwu and P. O. Asagba. (2017). *Security concept in web database development and administration — A review perspective*. IEEE 3rd International Conference on Electro-Technology for National Development (NIGERCON), Owerri, 2017, pp. 383-391.
- doi: 10.1109/NIGERCON.2017.8281910.