Warning

concerning copyright restrictions

The Copyright law of the United states (Title 17, United States Code) governs the making of photocopies or other reproductions of copyrighted material.

Under certain conditions specified in the law, libraries and archives are authorized to furnish a photocopy or reproduction.

One of three specified conditions is that the photocopy or reproduction is not to be used for any purpose other than private study, scholarship, or research.

If electronic transmission of reserve material is used for purposes in excess of what constitutes “fair use”, that user may be liable for copyright infringement.

This policy is in effect for the following document:

Thomas M. Connolly, Carolyn E. Begg

Database Architectures and the Web (Chapter 3) / from DataBase Systems: A Practical Approach to Design, Implementation and Management
Boston; London: Addison-Wesley, 2010. pp. 57- 87.

No further transmission or distribution of this material is permitted.

DATABASE SYSTEMS

A Practical Approach to Design, Implementation, and Management

FIFTH EDITION

THOMAS M. CONNOLLY | CAROLYN E. BEGG

UNIVERSITY OF THE WEST OF SCOTLAND

Addison-Wesley

Boston San Francisco New York
London Toronto Sydney Tokyo Singapore Madrid
Mexico City Munich Paris Cape Town Hong Kong Montreal

Editor-in-Chief	Michael Hirsch	Marketing Coordinator	Kathryn Ferranti
Editorial Assistant	Stephanie Sellinger	Senior Manufacturing Buyer	Carol Melville
Managing Editor	Jeffrey Holcomb	Composition	Aptara®, Inc.
Production Supervisor	Heather McNally	Production Coordination	Shelley Creager, Aptara®, Inc.
Art Director	Linda Knowles	Copyediting	Nancy Kotary
Cover Designer	Beth Paquin, Joyce Cosentino Wells	Proofreading	Don Smith
		Indexing	Steve Rath
Online Product Manager	Bethany Tidd	Text Design	Susan Raymond
Marketing Manager	Erin Davis		

Access the latest information about Addison-Wesley Computer Science titles from our World Wide Web site: http://www.pearsonhighered.com/cs

This interior of this book was composed in QuarkXpress 6.5.

Library of Congress Cataloging-in-Publication Data

Connolly, Thomas M.
Database systems : a practical approach to design, implementation, and management / Thomas M. Connolly, Carolyn E. Begg.—5th ed.
p. cm.
Includes bibliographical references and index.
ISBN-13: 978-0-321-52306-8
ISBN-10: 0-321-52306-7
1. Database design. 2. Database management. I. Begg, Carolyn E. II. Title.
QA76.9.D26C648 2010
005.74—dc22

2008055021

Addison-Wesley
is an imprint of

PEARSON

www.pearsonhighered.com

ISBN-13: 978-0-321-52306-8
ISBN-10: 0-321-52306-7

1 2 3 4 5 6 7 8 9 10—EB—13 12 11 10 09

CHAPTER

3 Database Architectures and the Web

Chapter Objectives

In this chapter you will learn:

- The meaning of the client–server architecture and the advantages of this type of architecture for a DBMS.
- The difference between two-tier, three-tier and *n*-tier client–server architectures.
- The function of an application server.
- The meaning of middleware and the different types of middleware that exist.
- The function and uses of Transaction Processing (TP) Monitors.
- The purpose of a Web service and the technological standards used to develop a Web service.
- The meaning of service-oriented architecture (SOA).
- The difference between distributed DBMSs, and distributed processing.
- The architecture of a data warehouse.
- The software components of a DBMS.
- About Oracle's logical and physical structure.

In Chapter 1, we provided a summary of the historical development of database systems from the 1960s onwards. During this period, although a better understanding of the functionality that users required was gained and new underlying data models were proposed and implemented, the software paradigm used to develop software systems more generally was undergoing significant change. Database system vendors had to recognize these changes and adapt to them to ensure that their systems were current with the latest thinking. In this chapter, we examine the different architectures that have been used and examine emerging developments in Web services and service-oriented architectures (SOA).

Structure of this Chapter In Section 3.1 we examine multi-user DBMS architectures focusing on the two-tier client–server architecture and the three-tier client–server architecture. We also examine the concept of middleware and discuss the different types of middleware that exist in the database field. In Section 3.2 we examine Web services which can be used to provide new types of business services to users, and SOA that promotes the design of loosely coupled and autonomous services that can be combined to provide more flexible composite business processes and applications. In Section 3.3 we briefly describe the architecture for a distributed DBMS that we consider in detail in Chapters 24 and 25, and distinguish it from distributed processing. In Section 3.4 we briefly describe the architecture for a data warehouse and associated tools that we consider in detail in Chapters 32–34. In Section 3.5 we examine an abstract internal architecture for a DBMS and in Section 3.6 we examine the logical and physical architecture of the Oracle DBMS.

3.1 Multi-user DBMS Architectures

In this section we look at the common architectures that are used to implement multi-user database management systems: teleprocessing, file-server, and client-server.

3.1.1 Teleprocessing

The traditional architecture for multi-user systems was teleprocessing, where there is one computer with a single central processing unit (CPU) and a number of terminals, as illustrated in Figure 3.1. All processing is performed within the boundaries of the same physical computer. User terminals are typically "dumb" ones, incapable of functioning on their own, and cabled to the central computer. The terminals send messages via the communications control subsystem of the operating system to the user's application program, which in turn uses the services of

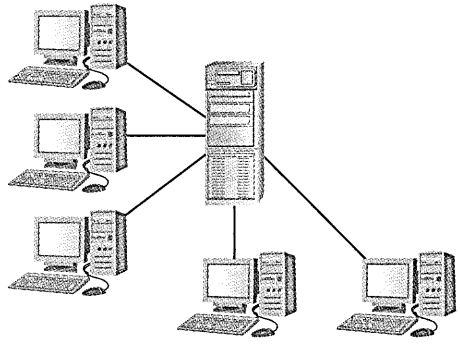

Figure 3.1 Teleprocessing topology.

the DBMS. In the same way, messages are routed back to the user's terminal. Unfortunately, this architecture placed a tremendous burden on the central computer, which had to not only run the application programs and the DBMS, but also carry out a significant amount of work on behalf of the terminals (such as formatting data for display on the screen).

In recent years, there have been significant advances in the development of high-performance personal computers and networks. There is now an identifiable trend in industry towards **downsizing**, that is, replacing expensive mainframe computers with more cost-effective networks of personal computers that achieve the same, or even better, results. This trend has given rise to the next two architectures: file-server and client-server.

3.1.2 File-Server Architecture

File server | A computer attached to a network with the primary purpose of providing shared storage for computer files such as documents, spreadsheets, images, and databases.

In a file-server environment, the processing is distributed about the network, typically a local area network (LAN). The file-server holds the files required by the applications and the DBMS. However, the applications and the DBMS run on each workstation, requesting files from the file-server when necessary, as illustrated in Figure 3.2. In this way, the file-server acts simply as a shared hard disk drive. The DBMS on each workstation sends requests to the file-server for all data that the DBMS requires that is stored on disk. This approach can generate a significant amount of network traffic, which can lead to performance problems. For example, consider a user request that requires the names of staff who work in the branch at 163 Main St. We can express this request in SQL (see Chapter 6) as:

```
SELECT fName, lName
FROM Branch b, Staff s
WHERE b.branchNo = s.branchNo AND b.street = '163 Main St';
```

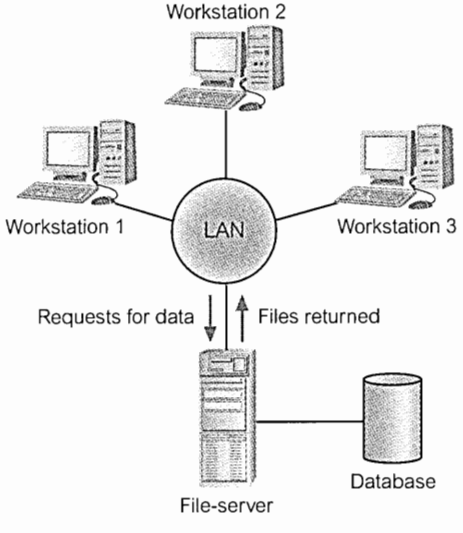

Figure 3.2 File-server architecture.

As the file-server has no knowledge of SQL, the DBMS must request the files corresponding to the Branch and Staff relations from the file-server, rather than just the staff names that satisfy the query.

The file-server architecture, therefore, has three main disadvantages:

(1) There is a large amount of network traffic.
(2) A full copy of the DBMS is required on each workstation.
(3) Concurrency, recovery, and integrity control are more complex, because there can be multiple DBMSs accessing the same files.

3.1.3 Traditional Two-Tier Client–Server Architecture

To overcome the disadvantages of the first two approaches and accommodate an increasingly decentralized business environment, the client–server architecture was developed. Client–server refers to the way in which software components interact to form a system. As the name suggests, there is a **client** process, which requires some resource, and a **server**, which provides the resource. There is no requirement that the client and server must reside on the same machine. In practice, it is quite common to place a server at one site in a LAN and the clients at the other sites. Figure 3.3 illustrates the client–server architecture and Figure 3.4 shows some possible combinations of the client–server topology.

Data-intensive business applications consist of four major components: the database, the transaction logic, the business and data application logic, and the user interface. The traditional two-tier client–server architecture provides a very basic separation of these components. The client (tier 1) is primarily responsible for the *presentation* of data to the user, and the server (tier 2) is primarily responsible for supplying *data services* to the client, as illustrated in Figure 3.5. Presentation services handle user interface actions and the main business and data application logic. Data services provide limited business application logic, typically validation

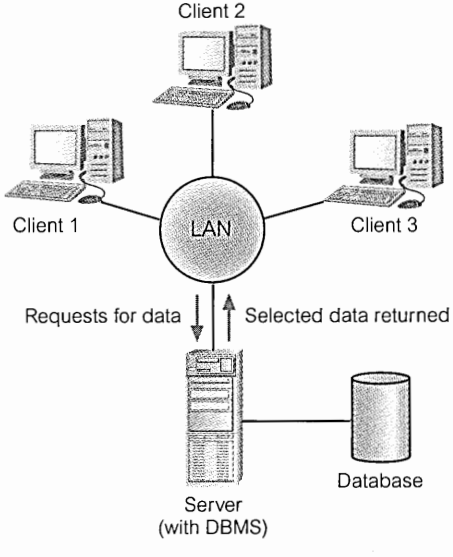

Figure 3.3 Client–server architecture.

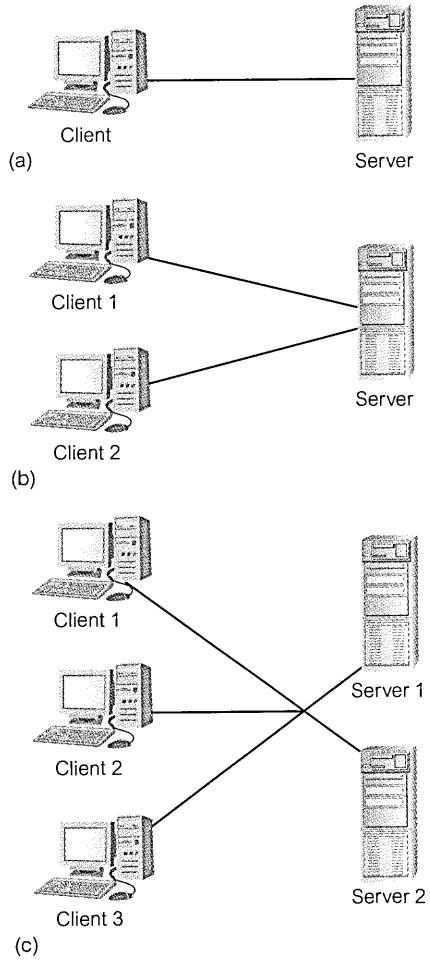

Figure 3.4
Alternative client–server topologies: (a) single client, single server; (b) multiple clients, single server; (c) multiple clients, multiple servers.

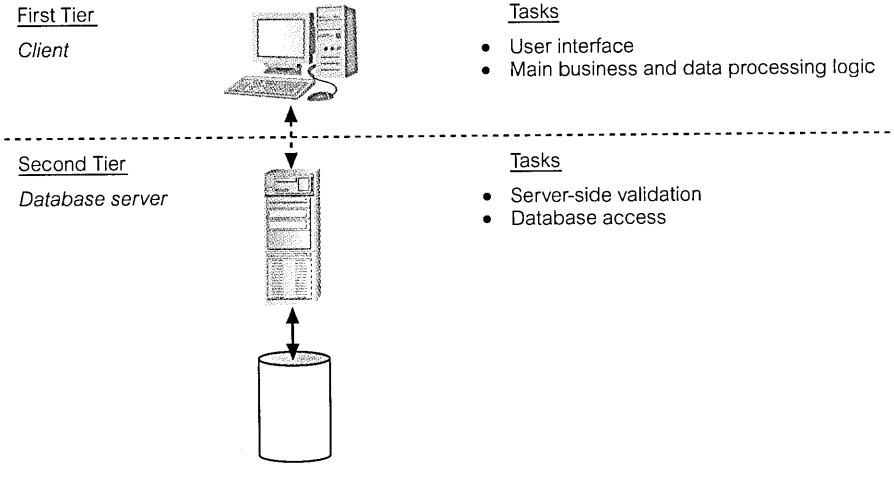

Figure 3.5
The traditional two-tier client–server architecture.

that the client is unable to carry out due to lack of information, and access to the requested data, independent of its location. The data can come from relational DBMSs, object-relational DBMSs, object-oriented DBMSs, legacy DBMSs, or proprietary data access systems. Typically, the client would run on end-user desktops and interact with a centralized database server over a network.

A typical interaction between client and server is as follows. The client takes the user's request, checks the syntax, and generates database requests in SQL or another database language appropriate to the application logic. It then transmits the message to the server, waits for a response, and formats the response for the end-user. The server accepts and processes the database requests, then transmits the results back to the client. The processing involves checking authorization, ensuring integrity, maintaining the system catalog, and performing query and update processing. In addition, it also provides concurrency and recovery control. The operations of client and server are summarized in Table 3.1.

There are many advantages to this type of architecture. For example:

- It enables wider access to existing databases.
- Increased performance: If the clients and server reside on different computers, then different CPUs can be processing applications in parallel. It should also be easier to tune the server machine if its only task is to perform database processing.
- Hardware costs may be reduced: It is only the server that requires storage and processing power sufficient to store and manage the database.
- Communication costs are reduced: Applications carry out part of the operations on the client and send only requests for database access across the network, resulting in less data being sent across the network.
- Increased consistency: The server can handle integrity checks, so that constraints need be defined and validated only in the one place, rather than having each application program perform its own checking.
- It maps on to open systems architecture quite naturally.

Some database vendors have used this architecture to indicate distributed database capability, that is, a collection of multiple, logically interrelated databases distributed over a computer network. However, although the client–server architecture can be used to provide distributed DBMSs, by itself it does not constitute a distributed DBMS. We discuss distributed DBMSs briefly in Section 3.3 and more fully in Chapters 24 and 25.

TABLE 3.1 Summary of client–server functions.

CLIENT	SERVER
Manages the user interface	Accepts and processes database requests from clients
Accepts and checks syntax of user input	Checks authorization
Processes application logic	Ensures integrity constraints not violated
Generates database requests and transmits to server	Performs query/update processing and transmits response to client
Passes response back to user	Maintains system catalog Provides concurrent database access Provides recovery control

3.1.4 Three-Tier Client–Server Architecture

The need for enterprise scalability challenged the traditional two-tier client–server model. In the mid 1990s, as applications became more complex and could potentially be deployed to hundreds or thousands of end-users, the client side presented two problems that prevented true scalability:

- A "fat" client, requiring considerable resources on the client's computer to run effectively. This includes disk space, RAM, and CPU power.
- A significant client-side administration overhead.

By 1995, a new variation of the traditional two-tier client–server model appeared to solve the problem of enterprise scalability. This new architecture proposed three layers, each potentially running on a different platform:

(1) The user interface layer, which runs on the end-user's computer (the *client*).
(2) The business logic and data processing layer. This middle tier runs on a server and is often called the *application server*.
(3) A DBMS, which stores the data required by the middle tier. This tier may run on a separate server called the *database server*.

As illustrated in Figure 3.6, the client is now responsible for only the application's user interface and perhaps performing some simple logic processing, such as input validation, thereby providing a "thin" client. The core business logic of the application now resides in its own layer, physically connected to the client and database server over a LAN or wide area network (WAN). One application server is designed to serve multiple clients.

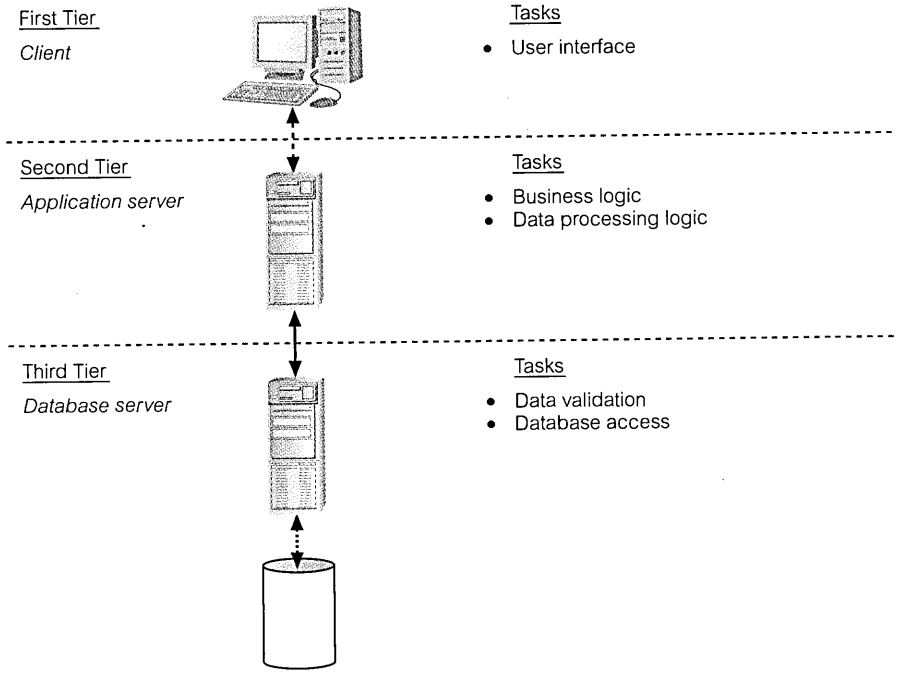

Figure 3.6
The three-tier architecture.

The three-tier design has many advantages over traditional two-tier or single-tier designs, which include:

- The need for less expensive hardware because the client is 'thin'.
- Application maintenance is centralized with the transfer of the business logic for many end-users into a single application server. This eliminates the concerns of software distribution that are problematic in the traditional two-tier client–server model.
- The added modularity makes it easier to modify or replace one tier without affecting the other tiers.
- Load balancing is easier with the separation of the core business logic from the database functions.

An additional advantage is that the three-tier architecture maps quite naturally to the Web environment, with a Web browser acting as the "thin" client, and a Web server acting as the application server.

3.1.5 *N*-Tier Architectures

The three-tier architecture can be expanded to n tiers, with additional tiers providing more flexibility and scalability. For example, as illustrated in Figure 3.7, the middle tier of the architecture could be split into two, with one tier for the Web

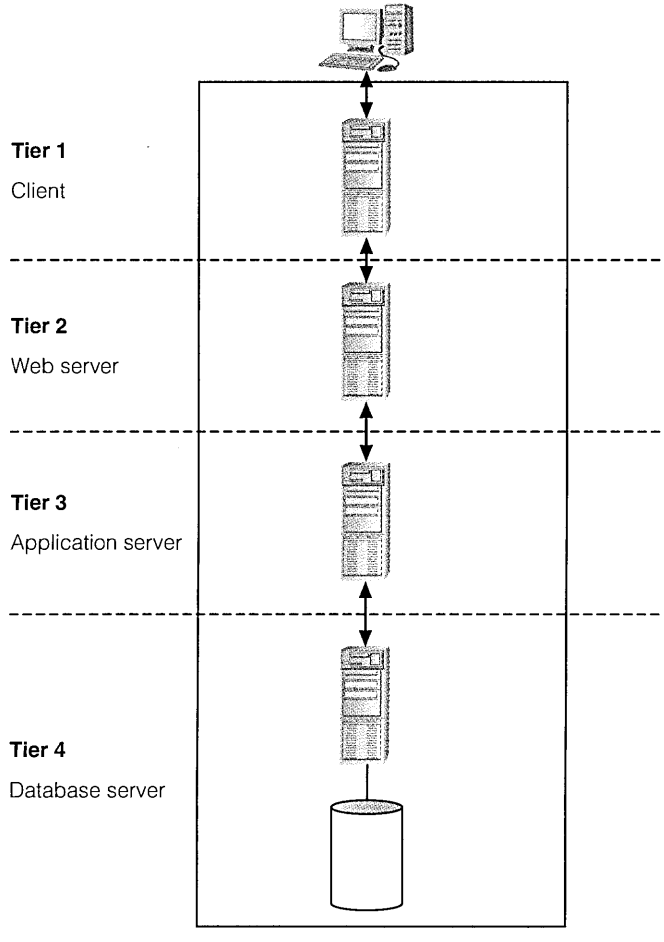

Figure 3.7 Four-tier architecture with the middle tier split into a Web server and application server.

server and another tier for the application server. In environments with a high volume of throughput, the single Web server could be replaced by a set of Web servers (or a *Web farm*) to achieve efficient load balancing.

Application servers

Application server | Hosts an application programming interface (API) to expose business logic and business processes for use by other applications.

An application server must handle a number of complex issues:

- concurrency;
- network connection management;
- providing access to all the database servers;
- database connection pooling;
- legacy database support;
- clustering support;
- load balancing;
- failover.

In Chapter 30 we will examine a number of application servers:

- Java Platform, Enterprise Edition (JEE), previously known as J2EE, is a specification for a platform for server programming in the Java programming language. As with other Java Community Process specifications, JEE is also considered informally to be a standard, as providers must agree to certain conformance requirements in order to declare their products to be "JEE-compliant." A JEE application server can handle the transactions, security, scalability, concurrency, and management of the components that are deployed to it, meaning that the developers should be able to concentrate more on the business logic of the components rather than on infrastructure and integration tasks.

  Some well known JEE application servers are WebLogic Server and OC4J (Oracle Containers for Java) from Oracle Corporation, JBoss from Red Hat, WebSphere Application Server from IBM, and the open source Glassfish Application Server, based on the commercial Sun Java System Application Server. We discuss the JEE platform and the technologies associated with accessing databases in Section 30.7.
- .NET Framework is Microsoft's offering for supporting the development of the middle tier. We discuss Microsoft .NET in Section 30.8.
- Oracle Application Server provides a set of services for assembling a scalable multitier infrastructure to support e-Business. We discuss the Oracle Application Server in Section 30.9.

3.1.6 Middleware

Middleware | Computer software that connects software components or applications.

Middleware is a generic term used to describe software that mediates with other software and allows for communication between disparate applications in a

heterogeneous system. The need for middleware arises when distributed systems become too complex to manage efficiently without a common interface. The need to make heterogeneous systems work efficiently across a network and be flexible enough to incorporate frequent modifications led to the development of middleware, which hides the underlying complexity of distributed systems.

Hurwitz (1998) defines six main types of middleware:

- *Asynchronous Remote Procedure Call (RPC):* An interprocess communication technology that allows a client to request a service in another address space (typically on another computer across a network) without waiting for a response. An RPC is initiated by the client sending a request message to a known remote server in order to execute a specified procedure using supplied parameters. This type of middleware tends to be highly scalable, as very little information about the connection and the session are maintained by either the client or the server. On the other hand, if the connection is broken, the client has to start over again from the beginning, so the protocol has low recoverability. Asynchronous RPC is most appropriate when transaction integrity is not required.
- *Synchronous RPC:* Similar to asynchronous RPC, however, while the server is processing the call, the client is blocked (it has to wait until the server has finished processing before resuming execution). This type of middleware is the least scalable but has the best recoverability.

  There are a number of analogous protocols to RPC, such as:

  - Java's Remote Method Invocation (Java RMI) API provides similar functionality to standard UNIX RPC methods;
  - XML-RPC is an RPC protocol that uses XML to encode its calls and HTTP as a transport mechanism. We will discuss HTTP in Chapter 30 and XML in Chapter 31.
  - Microsoft .NET Remoting offers RPC facilities for distributed systems implemented on the Windows platform. We will discuss the .NET platform in Section 30.8.
  - CORBA provides remote procedure invocation through an intermediate layer called the "Object Request Broker." We will discuss CORBA in Section 28.1.
  - The Thrift protocol and framework for the social networking Web site Facebook.
- *Publish/subscribe:* An asynchronous messaging protocol where *subscribers* subscribe to messages produced by *publishers*. Messages can be categorized into classes and subscribers express interest in one or more classes, and receive only messages that are of interest, without knowledge of what (if any) publishers there are. This decoupling of publishers and subscribers allows for greater scalability and a more dynamic network topology. Examples of publish/subscribe middleware include TIBCO Rendezvous from TIBCO Software Inc. and Ice (Internet Communications Engine) from ZeroC Inc.
- *Message-oriented middleware (MOM):* Software that resides on both the client and server and typically supports asynchronous calls between the client and server applications. Message queues provide temporary storage when the destination application is busy or not connected. There are many MOM products on the market, including WebSphere MS from IBM, MSMQ (Microsoft Message Queue Server), Sun JMS (Java Messaging Service), which is part of JEE and enables the

development of portable, message-based applications in Java, Sun Java System Message Queue (SJSMQ), which implements JMS, and MessageQ from BEA Systems, now a subsidiary of Oracle Corporation.

- *Object-request broker (ORB):* Manages communication and data exchange between objects. ORBs promote interoperability of distributed object systems by allowing developers to build systems by integrating together objects, possibly from different vendors, that communicate with each other via the ORB. The Common Object Requesting Broker Architecture (CORBA) is a standard defined by the Object Management Group (OMG) that enables software components written in multiple computer languages and running on multiple computers to work together. An example of a commercial ORB middleware product is Orbix from IONA Technologies.
- *SQL-oriented data access:* Connects applications with databases across the network and translates SQL requests into the database's native SQL or other database language. SQL-oriented middleware eliminates the need to code SQL-specific calls for each database and to code the underlying communications. More generally, *database-oriented middleware* connects applications to any type of database (not necessarily a relational DBMS through SQL). Examples include Microsoft's ODBC (Open Database Connectivity) API, which exposes a single interface to facilitate access to a database and then uses drivers to accommodate differences between databases, and the JDBC API, which uses a single set of Java methods to facilitate access to multiple databases. Within this category we would also include *gateways,* which act as mediators in distributed DBMSs to translate one database language or dialect of a language into to another language or dialect (for example, Oracle SQL into IBM's DB2 SQL or Microsoft SQL Server SQL into Object Query Language, or OQL). We will discuss gateways in Chapter 24 and ODBC and JDBC in Chapter 30.

In the next section, we examine one particular type of middleware for transaction processing.

3.1.7 Transaction Processing Monitors

TP Monitor

A program that controls data transfer between clients and servers in order to provide a consistent environment, particularly for online transaction processing (OLTP).

Complex applications are often built on top of several **resource managers** (such as DBMSs, operating systems, user interfaces, and messaging software). A Transaction Processing Monitor, or TP Monitor, is a middleware component that provides access to the services of a number of resource managers and provides a uniform interface for programmers who are developing transactional software. A TP Monitor forms the middle tier of a three-tier architecture, as illustrated in Figure 3.8. TP Monitors provide significant advantages, including:

- *Transaction routing:* The TP Monitor can increase scalability by directing transactions to specific DBMSs.
- *Managing distributed transactions:* The TP Monitor can manage transactions that require access to data held in multiple, possibly heterogeneous, DBMSs. For example, a transaction may require to update data items held in an Oracle

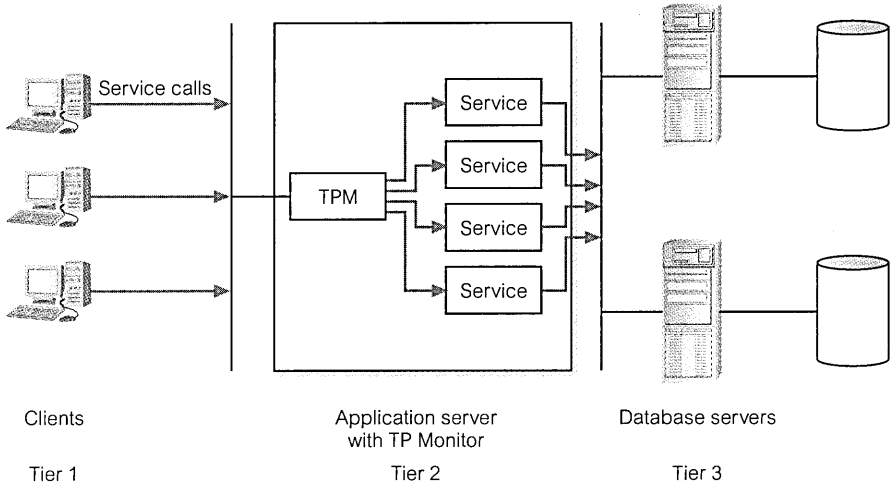

Figure 3.8 The Transaction Processing Monitor as the middle tier of a three-tier client–server architecture.

DBMS at site 1, an Informix DBMS at site 2, and an IMS DBMS as site 3. TP Monitors normally control transactions using the X/Open Distributed Transaction Processing (DTP) standard. A DBMS that supports this standard can function as a resource manager under the control of a TP Monitor acting as a transaction manager. We discuss distributed transactions and the DTP standard in Chapters 23 and 24.

- *Load balancing:* The TP Monitor can balance client requests across multiple DBMSs on one or more computers by directing client service calls to the least loaded server. In addition, it can dynamically bring in additional DBMSs as required to provide the necessary performance.
- *Funneling:* In environments with a large number of users, it may sometimes be difficult for all users to be logged on simultaneously to the DBMS. In many cases, we would find that users generally do not need continuous access to the DBMS. Instead of each user connecting to the DBMS, the TP Monitor can establish connections with the DBMSs as and when required, and can funnel user requests through these connections. This allows a larger number of users to access the available DBMSs with a potentially much smaller number of connections, which in turn would mean less resource usage.
- *Increased reliability:* The TP Monitor acts as a *transaction manager*, performing the necessary actions to maintain the consistency of the database, with the DBMS acting as a *resource manager*. If the DBMS fails, the TP Monitor may be able to resubmit the transaction to another DBMS or can hold the transaction until the DBMS becomes available again.

TP Monitors are typically used in environments with a very high volume of transactions, where the TP Monitor can be used to offload processes from the DBMS server. Prominent examples of TP Monitors include CICS and Encina from IBM (which are primarily used on IBM AIX or Windows NT and bundled now in the IBM TXSeries) and Tuxedo from BEA Systems.

3.2 Web Services and Service-Oriented Architectures

3.2.1 Web Services

Web service | A software system designed to support interoperable machine-to-machine interaction over a network.

Although it has been only about 20 years since the conception of the Internet, in this relatively short period of time it has profoundly changed many aspects of society, including business, government, broadcasting, shopping, leisure, communication, education, and training. Though the Internet has allowed companies to provide a wide range of services to users, sometimes called B2C (Business to Consumer), Web services allow applications to integrate with other applications across the Internet and may be a key technology that supports B2B (Business to Business) interaction.

Unlike other Web-based applications, Web services have no user interface and are not aimed at Web browsers. Web services instead share business logic, data, and processes through a programmatic interface across a network. In this way, it is the applications that interface and not the users. Developers can then add the Web service to a Web page (or an executable program) to offer specific functionality to users. Examples of Web services include:

- Microsoft Virtual Earth Web services offer static map images (gif, jpeg, and png), direct map tile access, search functionality, geocoding, reverse geocoding, and routing. Microsoft MapPoint Web service provides access to location-based services, such as maps, driving directions, and proximity searches.
- Amazon S3 is a simple Web services interface that can be used to store and retrieve large amounts of data, at any time, from anywhere on the Web. It gives any developer access to the same highly scalable, reliable, fast, inexpensive data storage infrastructure that Amazon uses to run its own global network of Web sites. Charges are based on the "pay-as-you-go" policy, currently $0.15 per GB for the first 50TB/month of storage used.
- Geonames provides a number of location-related Web services; for example, to return a set of Wikipedia entries as XML documents for a given place name or to return the time zone for a given latitude/longitude.
- DOTS Web services from ObjectServices, an early adopter of Web services, provide a range of services such as company information, reverse telephone number lookup, email address validation, weather information, IP address-to-location determination.

Key to the Web services approach is the use of widely accepted technologies and standards, such as:

- XML (extensible Markup Language).
- SOAP (Simple Object Access Protocol) is a communication protocol for exchanging structured information over the Internet and uses a message format based on XML. It is both platform- and language-independent.
- WSDL (Web Services Description Language) protocol, again based on XML, is used to describe and locate a Web service.

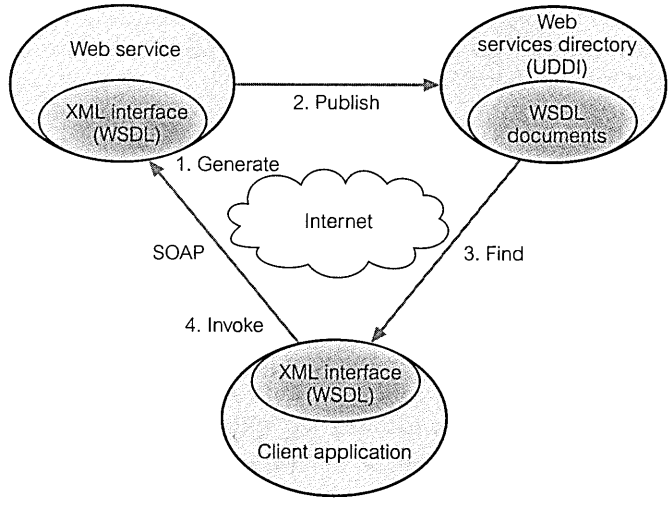

Figure 3.9
Relationship between WSDL, UDDI, and SOAP.

- UDDI (Universal Discovery, Description, and Integration) protocol is a platform-independent, XML-based registry for businesses to list themselves on the Internet. It was designed to be interrogated by SOAP messages and to provide access to WSDL documents describing the protocol bindings and message formats required to interact with the Web services listed in its directory.

Figure 3.9 illustrates the relationship between these technologies. From the database perspective, Web services can be used both from within the database (to invoke an external Web service as a *consumer)* and the Web service itself can access its own database (as a *provider)* to maintain the data required to provide the requested service. We will discuss Web services in Section 30.2.5 and SOAP, WSDL, and UDDI in Section 31.3.

3.2.2 Service-Oriented Architectures (SOA)

SOA

A business-centric software architecture for building applications that implement business processes as sets of services published at a granularity relevant to the service consumer. Services can be invoked, published, and discovered, and are abstracted away from the implementation using a single standards-based form of interface.

Flexibility is recognized as a key requirement for businesses in a time when IT is providing business opportunities that were never envisaged in the past while at the same time the underlying technologies are rapidly changing. Reusability has often been seen as a major goal of software development and underpins the object-oriented paradigm: object-oriented programming (OOP) may be viewed as a collection of cooperating *objects*, as opposed to a traditional view in which a program may be seen as a group of tasks to compute. In OOP, each object is capable of receiving messages, processing data, and sending messages to other objects. In traditional IT architectures, business process activities, applications, and data tend to be locked in independent, often-incompatible "silos," where users have to navigate separate

networks, applications, and databases to conduct the chain of activities that complete a business process. Unfortunately, independent silos absorb an inordinate amount of IT budget and staff time to maintain. This architecture is illustrated in Figure 3.10(a) where we have three processes: Service Scheduling, Order Processing, and Account Management, each accessing a number of databases. Clearly there are common "services" in the activities to be performed by these processes. If the business requirements change or new opportunities present themselves, the lack of independence among these processes may lead to difficulties in quickly adapting these processes.

The SOA approach attempts to overcome this difficulty by designing loosely coupled and autonomous *services* that can be combined to provide flexible composite business processes and applications. Figure 3.10(b) illustrates the same business processes rearchitected to use a service-oriented approach.

The essence of a service, therefore, is that the provision of the service is independent of the application using the service. Service providers can develop specialized services and offer these to a range of service users from different organizations. Applications may be constructed by linking services from various providers using either a standard programming language or a specialized service orchestration language such as BPEL (Business Process Execution Language).

What makes Web services designed for SOA different from other Web services is that they typically follow a number of distinct conventions. The following are a

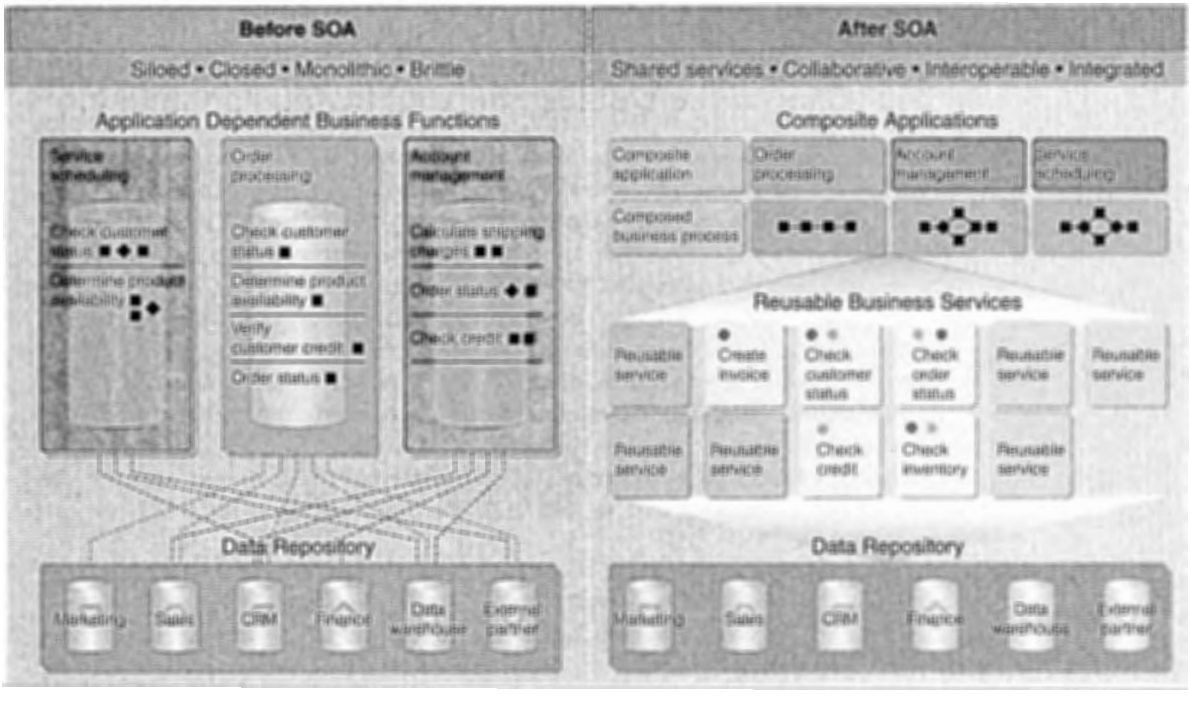

Figure 3.10 (a) Traditional IT architecture for three business processes; (b) service-oriented architecture that splits the processes into a number of reusable services.

set of common SOA principles that provide a unique design approach for building Web services for SOA:

- *loose coupling:* services must be designed to interact on a loosely coupled basis;
- *reusability:* logic that can potentially be reused is designed as a separate service;
- *contract:* services adhere to a communications contract that defines the information exchange and any additional service description information, specified by one or more service description documents;
- *abstraction:* beyond what is described in the service contract, services hide logic from the outside world;
- *composability:* services may compose other services, so that logic can be represented at different levels of granularity thereby promoting reusability and the creation of abstraction layers;
- *autonomy:* services have control over the logic they encapsulate and are not dependent upon other services to execute this governance;
- *stateless:* services should not be required to manage state information, as this can affect their ability to remain loosely-coupled;
- *discoverability:* services are designed to be outwardly descriptive so that they can be found and assessed via available discovery mechanisms.

Note that SOA is not restricted to Web services and could be used with other technologies. Further discussion of SOA is beyond the scope of this book and the interested reader is referred to the additional reading listed at the end of the book for this chapter.

3.3 Distributed DBMSs

As discussed in Chapter 1, a major motivation behind the development of database systems is the desire to integrate the operational data of an organization and to provide controlled access to the data. Although we may think that integration and controlled access implies centralization, this is not the intention. In fact, the development of computer networks promotes a decentralized mode of work. This decentralized approach mirrors the organizational structure of many companies, which are logically distributed into divisions, departments, projects, and so on, and physically distributed into offices, plants, or factories, where each unit maintains its own operational data. The development of a distributed DBMS that reflects this organizational structure, makes the data in all units accessible, and stores data proximate to the location where it is most frequently used, should improve the ability to share the data and should improve the efficiency with which we can access the data.

Distributed database | A logically interrelated collection of shared data (and a description of this data), physically distributed over a computer network.

Distributed DBMS | The software system that permits the management of the distributed database and makes the distribution transparent to users.

A **distributed database management system** (**DDBMS**) consists of a single logical database that is split into a number *of fragments*. Each fragment is stored on one or more computers (*replicas*) under the control of a separate DBMS, with the computers connected by a communications network. Each site is capable of independently processing user requests that require access to local data (that is, each site has some degree of local autonomy) and is also capable of processing data stored on other computers in the network.

Users access the distributed database via applications. Applications are classified as those that do not require data from other sites (*local applications*) and those that do require data from other sites (*global applications*). We require a DDBMS to have at least one global application. A DDBMS therefore has the following characteristics:

- a collection of logically related shared data;
- data split into a number of fragments;
- fragments may be replicated;
- fragments/replicas are allocated to sites;
- sites are linked by a communications network;
- data at each site is under the control of a DBMS;
- DBMS at each site can handle local applications, autonomously;
- each DBMS participates in at least one global application.

It is not necessary for every site in the system to have its own local database, as illustrated by the topology of the DDBMS shown in Figure 3.11.

From the definition of the DDBMS, the system is expected to make the distribution *transparent* (invisible) to the user. Thus, the fact that a distributed database is split into fragments that can be stored on different computers and perhaps replicated should be hidden from the user. The objective of transparency is to make the

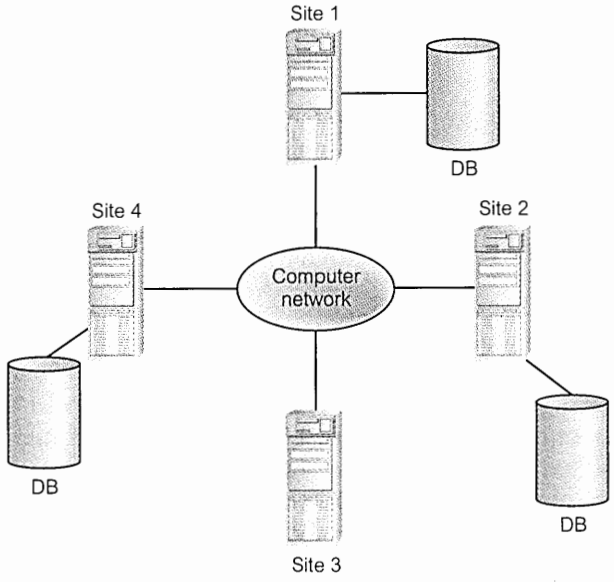

Figure 3.11 Distributed database management system.

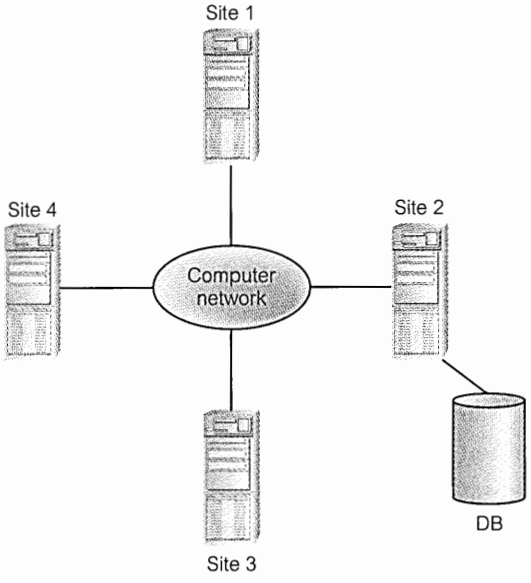

Figure 3.12 Distributed processing.

distributed system appear like a centralized system. This is sometimes referred to as the *fundamental principle* of distributed DBMSs. This requirement provides significant functionality for the end-user but, unfortunately, creates many additional problems that have to be handled by the DDBMS.

Distributed processing

It is important to make a distinction between a distributed DBMS and distributed processing:

Distributed processing	A centralized database that can be accessed over a computer network.

The key point with the definition of a distributed DBMS is that the system consists of data that is physically distributed across a number of sites in the network. If the data is centralized, even though other users may be accessing the data over the network, we do not consider this to be a distributed DBMS simply distributed processing. We illustrate the topology of distributed processing in Figure 3.12. Compare this figure, which has a central database at site 2, with Figure 3.11, which shows several sites each with their own database. We will discuss distributed DBMSs in depth in Chapters 24 and 25.

3.4 Data Warehousing

Since the 1970s, organizations have largely focused their investment in new computer systems (called online transaction processing or OLTP systems) that automate business processes. In this way, organizations gained competitive advantage through systems that offered more efficient and cost-effective services to the customer.

Throughout this period, organizations accumulated growing amounts of data stored in their operational databases. However, in recent times, where such systems are commonplace, organizations are focusing on ways to use operational data to support decision making as a means of regaining competitive advantage.

Operational systems were never primarily designed to support business decision making and so using such systems may never be an easy solution. The legacy is that a typical organization may have numerous operational systems with overlapping and sometimes contradictory definitions, such as data types. The challenge for an organization is to turn its archives of data into a source of knowledge, so that a single integrated/consolidated view of the organization's data is presented to the user. The concept of a **data warehouse** was deemed the solution to meet the requirements of a system capable of supporting decision making, receiving data from multiple operational data sources.

Data warehouse

A consolidated/integrated view of corporate data drawn from disparate operational data sources and a range of end-user access tools capable of supporting simple to highly complex queries to support decision making.

The data held in a data warehouse is described as being subject-oriented, integrated, time-variant, and nonvolatile (Inmon, 1993).

- *Subject-oriented*, as the warehouse is organized around the major subjects of the organization (such as customers, products, and sales) rather than the major application areas (such as customer invoicing, stock control, and product sales). This is reflected in the need to store decision-support data rather than application-oriented data.
- *Integrated*, because of the coming together of source data from different organization-wide applications systems. The source data is often inconsistent, using, for example, different data types and/or formats. The integrated data source must be made consistent to present a unified view of the data to the users.
- *Time-variant*, because data in the warehouse is accurate and valid only at some point in time or over some time interval.
- *Nonvolatile*, as the data is not updated in real time but is refreshed from operational systems on a regular basis. New data is always added as a supplement to the database, rather than a replacement.

The typical architecture of a data warehouse is shown in Figure 3.13.

The source of *operational data* for the data warehouse is supplied from mainframes, proprietary file systems, private workstations and servers, and external systems such as the Internet. An *operational data store* (ODS) is a repository of current and integrated operational data used for analysis. It is often structured and supplied with data in the same way as the data warehouse, but may in fact act simply as a staging area for data to be moved into the warehouse. The *load manager* performs all the operations associated with the extraction and loading of data into the warehouse. The *warehouse manager* performs all the operations associated with the management of the data, such as the transformation and merging of source data; creation of indexes and views on base tables; generation of aggregations, and backing up and

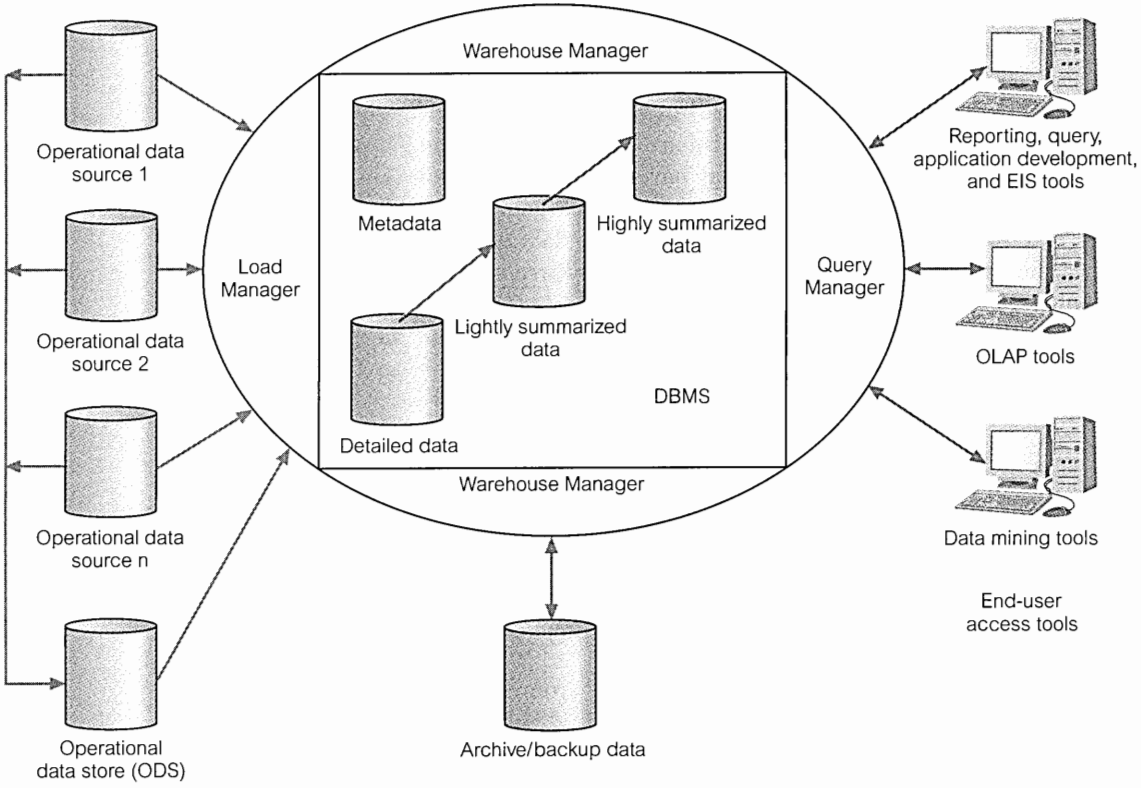

Figure 3.13 The typical architecture of a data warehouse.

archiving data. The *query manager* performs all the operations associated with the management of user queries. *Detailed data* is not stored online but is made available by summarizing the data to the next level of detail. However, on a regular basis, detailed data is added to the warehouse to supplement the summarized data. The warehouse stores all the predefined *lightly and highly summarized data* generated by the warehouse manager. The purpose of summary information is to speed up the performance of queries. Although there are increased operational costs associated with initially summarizing the data, this cost is offset by removing the requirement to continually perform summary operations (such as sorting or grouping) in answering user queries. The summary data is updated continuously as new data is loaded into the warehouse. Detailed and summarized data is stored offline for the purposes of archiving and backup. *Metadata* (data about data) definitions are used by all the processes in the warehouse, including the extraction and loading processes; the warehouse management process; and as part of the query management process.

The principal purpose of data warehousing is to provide information to business users for strategic decision making. These users interact with the warehouse using *end-user access tools*. The data warehouse must efficiently support *ad hoc* and routine analysis as well as more complex data analysis. The types of end-user access tools

typically include reporting and query tools, application development tools, executive information system (EIS) tools, online analytical processing (OLAP) tools, and data mining tools. We will discuss data warehousing, OLAP, and data mining tools in depth in Chapters 32–35.

3.5 Components of a DBMS

DBMSs are highly complex and sophisticated pieces of software that aim to provide the services discussed in Section 2.4. It is not possible to generalize the component structure of a DBMS, as it varies greatly from system to system. However, it is useful when trying to understand database systems to try to view the components and the relationships between them. In this section, we present a possible architecture for a DBMS. We examine the architecture of the Oracle DBMS in the next section.

A DBMS is partitioned into several software components (or *modules*), each of which is assigned a specific operation. As stated previously, some of the functions of the DBMS are supported by the underlying operating system. However, the operating system provides only basic services and the DBMS must be built on top of it. Thus, the design of a DBMS must take into account the interface between the DBMS and the operating system.

The major software components in a DBMS environment are depicted in Figure 3.14. This diagram shows how the DBMS interfaces with other software

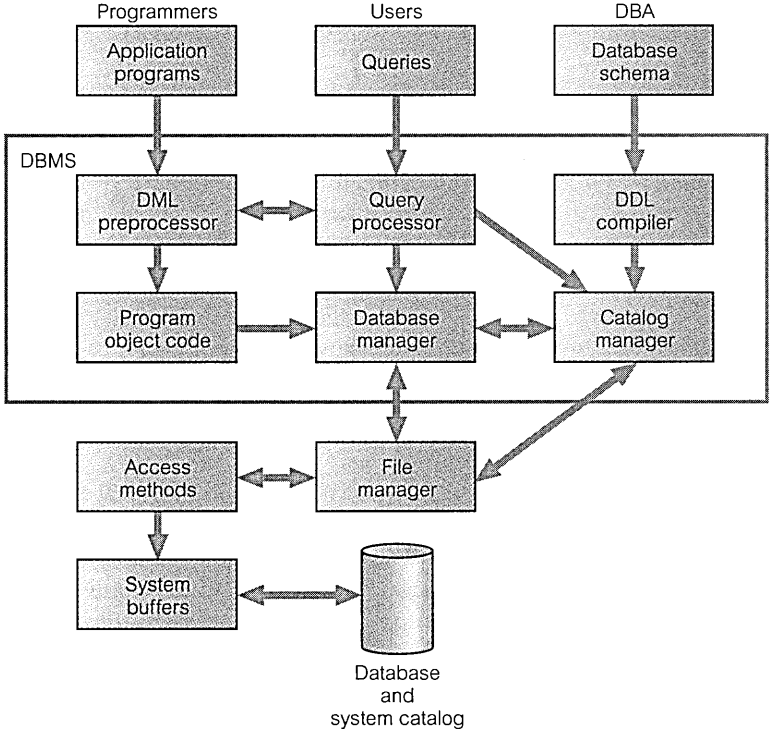

Figure 3.14
Major components of a DBMS.

components, such as user queries and access methods (file management techniques for storing and retrieving data records). We will provide an overview of file organizations and access methods in Appendix F. For a more comprehensive treatment, the interested reader is referred to Teorey and Fry (1982), Weiderhold (1983), Smith and Barnes (1987), and Ullman (1988).

Figure 3.14 shows the following components:

- *Query processor.* This is a major DBMS component that transforms queries into a series of low-level instructions directed to the database manager. We discuss query processing in Chapter 23.
- *Database manager (DM).* The DM interfaces with user-submitted application programs and queries. The DM accepts queries and examines the external and conceptual schemas to determine what conceptual records are required to satisfy the request. The DM then places a call to the file manager to perform the request. The components of the DM are shown in Figure 3.15.

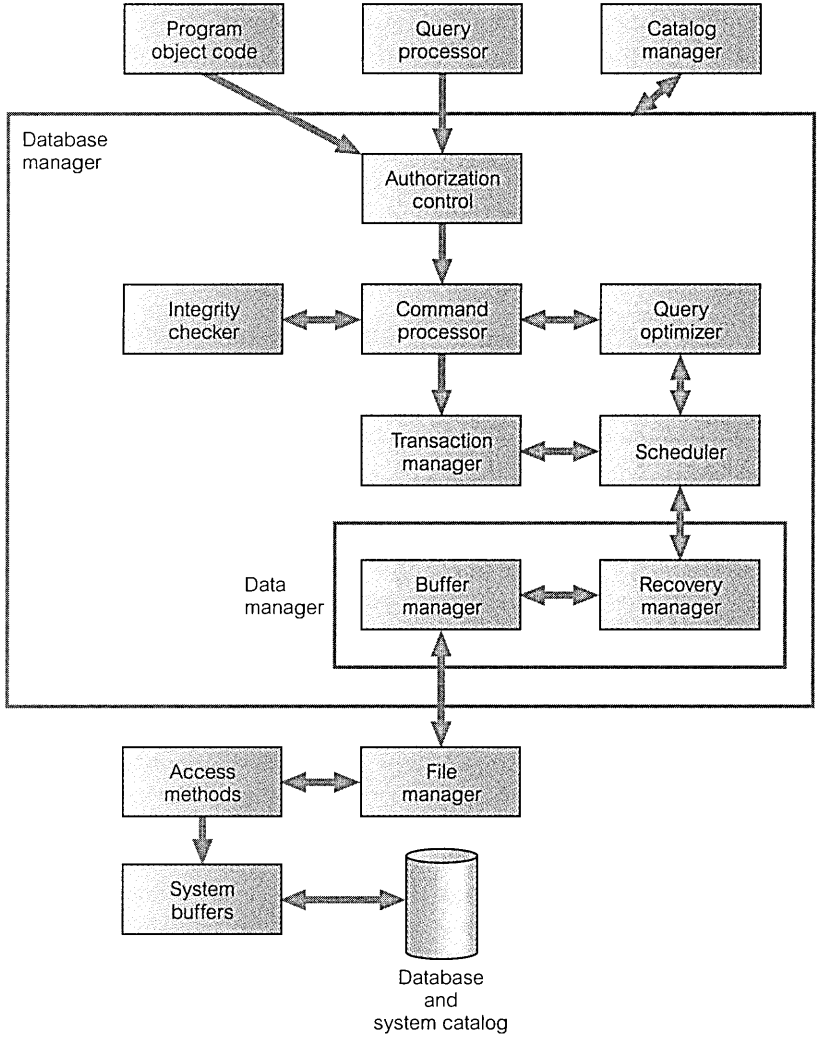

Figure 3.15 Components of a database manager.

- *File manager*. The file manager manipulates the underlying storage files and manages the allocation of storage space on disk. It establishes and maintains the list of structures and indexes defined in the internal schema. If hashed files are used, it calls on the hashing functions to generate record addresses. However, the file manager does not directly manage the physical input and output of data. Rather, it passes the requests on to the appropriate access methods, which either read data from or write data into the system buffer (or *cache*).
- *DML preprocessor*. This module converts DML statements embedded in an application program into standard function calls in the host language. The DML preprocessor must interact with the query processor to generate the appropriate code.
- *DDL compiler*. The DDL compiler converts DDL statements into a set of tables containing metadata. These tables are then stored in the system catalog while control information is stored in data file headers.
- *Catalog manager*. The catalog manager manages access to and maintains the system catalog. The system catalog is accessed by most DBMS components.

The major software components for the *database manager* are as follows:

- *Authorization control*. This module confirms whether the user has the necessary authorization to carry out the required operation.
- *Command processor*. Once the system has confirmed that the user has authority to carry out the operation, control is passed to the command processor.
- *Integrity checker*. For an operation that changes the database, the integrity checker checks whether the requested operation satisfies all necessary integrity constraints (such as key constraints).
- *Query optimizer*. This module determines an optimal strategy for the query execution. We discuss query optimization in Chapter 23.
- *Transaction manager*. This module performs the required processing of operations that it receives from transactions.
- *Scheduler*. This module is responsible for ensuring that concurrent operations on the database proceed without conflicting with one another. It controls the relative order in which transaction operations are executed.
- *Recovery manager*. This module ensures that the database remains in a consistent state in the presence of failures. It is responsible for transaction commit and abort.
- *Buffer manager*. This module is responsible for the transfer of data between main memory and secondary storage, such as disk and tape. The recovery manager and the buffer manager are sometimes referred to collectively as the *data manager*. The buffer manager is sometimes known as the *cache manager*.

We discuss the last four modules in Chapter 22. In addition to the previously mentioned modules, several other data structures are required as part of the physical-level implementation. These structures include data and index files and the system catalog. An attempt has been made to standardize DBMSs, and a reference model was proposed by the Database Architecture Framework Task Group (DAFTG, 1986). The purpose of this reference model was to define a conceptual framework aiming to divide standardization attempts into manageable pieces and to show at a very broad level how these pieces could be interrelated.

3.6 Oracle Architecture

Oracle is based on the client–server architecture examined in Section 3.1.3. The Oracle server consists of the *database* (the raw data, including log and control files) and the *instance* (the processes and system memory on the server that provide access to the database). An instance can connect to only one database. The database consists of a *logical structure*, such as the database schema, and a *physical structure*, containing the files that make up an Oracle database. We now discuss the logical and physical structure of the database and the system processes in more detail.

3.6.1 Oracle's Logical Database Structure

At the logical level, Oracle maintains *tablespaces*, *schemas*, and *data blocks* and *extents/segments*.

Tablespaces

An Oracle database is divided into logical storage units called **tablespaces**. A tablespace is used to group related logical structures together. For example, tablespaces commonly group all the application's objects to simplify some administrative operations.

Every Oracle database contains a tablespace named SYSTEM, which is created automatically when the database is created. The SYSTEM tablespace always contains the system catalog tables (called the *data dictionary* in Oracle) for the entire database. A small database might need only the SYSTEM tablespace; however, it is recommended that at least one additional tablespace is created to store user data separate from the data dictionary, thereby reducing contention among dictionary objects and schema objects for the same datafiles (see Figure 19.11). Figure 3.16 illustrates an Oracle database consisting of the SYSTEM tablespace and a USER_DATA tablespace.

A new tablespace can be created using the CREATE TABLESPACE command, for example:

```
CREATE TABLESPACE user_data
DATAFILE 'DATA3.ORA' SIZE 100K
EXTENT MANAGEMENT LOCAL
SEGMENT SPACE MANAGEMENT AUTO;
```

A table can then be associated with a specific tablespace using the CREATE TABLE or ALTER TABLE statement, for example:

```
CREATE TABLE PropertyForRent (propertyNo VARCHAR2(5) NOT NULL, . . . )
TABLESPACE user_data;
```

If no tablespace is specified when creating a new table, the default tablespace associated with the user when the user account was set up is used.

Users, schemas, and schema objects

A **user** (sometimes called a **username**) is a name defined in the database that can connect to, and access, objects. A **schema** is a named collection of schema objects,

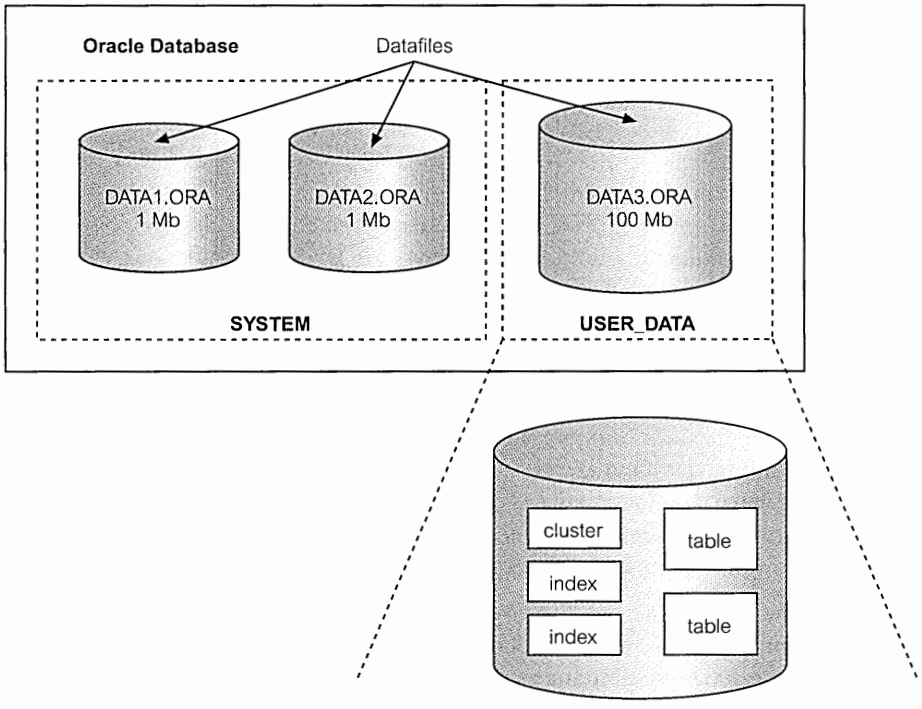

Figure 3.16 Relationship between an Oracle database, tablespaces, and datafiles.

such as tables, views, indexes, clusters, and procedures, associated with a particular user. Schemas and users help DBAs manage database security.

To access a database, a user must run a database application (such as Oracle Forms or SQL*Plus) and connect using a username defined in the database. When a database user is created, a corresponding schema of the same name is created for the user. By default, once a user connects to a database, the user has access to all objects contained in the corresponding schema. As a user is associated only with the schema of the same name; the terms "user" and "schema" are often used interchangeably. (Note that there is no relationship between a tablespace and a schema: objects in the same schema can be in different tablespaces, and a tablespace can hold objects from different schemas.)

Data blocks, extents, and segments

The **data block** is the smallest unit of storage that Oracle can use or allocate. One data block corresponds to a specific number of bytes of physical disk space. The data block size can be set for each Oracle database when it is created. This data block size should be a multiple of the operating system's block size (within the system's maximum operating limit) to avoid unnecessary I/O. A data block has the following structure:

- *Header:* Contains general information such as block address and type of segment.
- *Table directory:* Contains information about the tables that have data in the data block.
- *Row directory:* Contains information about the rows in the data block.
- *Row data:* Contains the actual rows of table data. A row can span blocks.

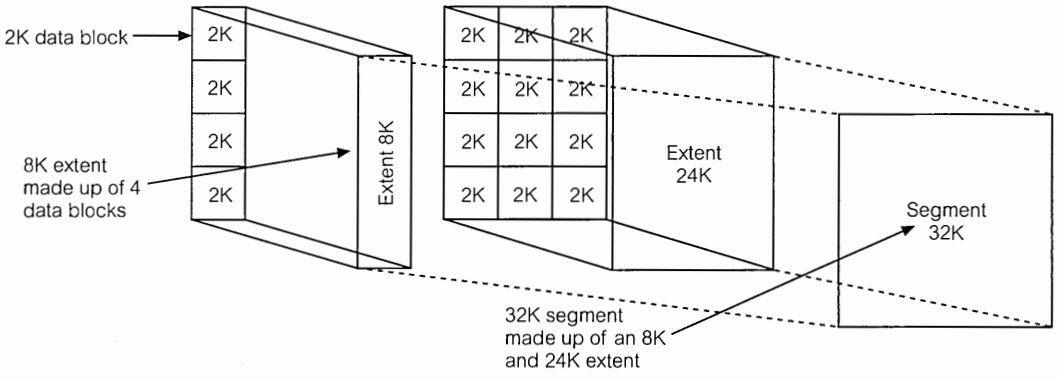

Figure 3.17 Relationship between Oracle data blocks, extents, and segments.

- *Free space:* Allocated for the insertion of new rows and updates to rows that require additional space. As of Oracle8*i*, Oracle can manage free space automatically, although there is an option to manage it manually.

We show how to estimate the size of an Oracle table using these components in Appendix J on the Web site for this book. The next level of logical database space is called an **extent**. An extent is a specific number of contiguous data blocks allocated for storing a specific type of information. The level above an extent is called a **segment**. A segment is a set of extents allocated for a certain logical structure. For example, each table's data is stored in its own data segment, and each index's data is stored in its own index segment. Figure 3.17 shows the relationship between data blocks, extents, and segments. Oracle dynamically allocates space when the existing extents of a segment become full. Because extents are allocated as needed, the extents of a segment may or may not be contiguous on disk.

3.6.2 Oracle's Physical Database Structure

The main physical database structures in Oracle are datafiles, redo log files, and control files.

Datafiles

Every Oracle database has one or more physical datafiles. The data of logical database structures (such as tables and indexes) is physically stored in these datafiles. As shown in Figure 3.16, one or more datafiles form a tablespace. The simplest Oracle database would have one tablespace and one datafile. A more complex database might have four tablespaces, each consisting of two datafiles, giving a total of eight datafiles.

Redo log files

Every Oracle database has a set of two or more redo log files that record all changes made to data for recovery purposes. Should a failure prevent modified

data from being permanently written to the datafiles, the changes can be obtained from the redo log, thus preventing work from being lost. We discuss recovery in detail in Section 22.3.

Control files

Every Oracle database has a control file that contains a list of all the other files that make up the database, such as the datafiles and redo log files. For added protection, it is recommended that the control file should be multiplexed (multiple copies may be written to multiple devices). Similarly, it may be advisable to multiplex the redo log files as well.

The Oracle instance

The Oracle instance consists of the Oracle processes and shared memory required to access information in the database. The instance is made up of the Oracle background processes, the user processes, and the shared memory used by these processes, as illustrated in Figure 3.18. Among other things, Oracle uses shared memory for caching data and indexes as well as storing shared program code. Shared memory is broken into various *memory structures*, of which the basic ones are the System Global Area (SGA) and the Program Global Area (PGA).

- *System global area.* The SGA is an area of shared memory that is used to store data and control information for one Oracle instance. The SGA is allocated when the Oracle instance starts and deallocated when the Oracle instance shuts down. The information in the SGA consists of the following memory structures, each of which has a fixed size and is created at instance startup:
  - *Database buffer cache.* This contains the most recently used data blocks from the database. These blocks can contain modified data that has not yet been written to disk (*dirty blocks*), blocks that have not been modified, or blocks that have been written to disk since modification (*clean blocks*). By storing the most recently used blocks, the most active buffers stay in memory to reduce I/O and improve performance. We discuss buffer management policies in Section 22.3.2.
  - *Redo log buffer.* This contains the redo log file entries, which are used for recovery purposes (see Section 22.3). The background process LGWR (explained shortly) writes the redo log buffer to the active online redo log file on disk.
  - *Shared pool.* This contains the shared memory structures, such as shared SQL areas in the library cache and internal information in the data dictionary. The shared SQL areas contain parse trees and execution plans for the SQL queries. If multiple applications issue the same SQL statement, each can access the shared SQL area to reduce the amount of memory needed and to reduce the processing time used for parsing and execution. We discuss query processing in Chapter 23.
- *Program global area.* The PGA is an area of shared memory that is used to store data and control information for the Oracle server processes. The size and content of the PGA depends on the Oracle server options installed.
- *User processes.* Each user process represents the user's connection to the Oracle server (for example, through SQL*Plus or an Oracle Forms application). The user process manipulates the user's input, communicates with the Oracle server process, displays the information requested by the user and, if required, processes this information into a more useful form.

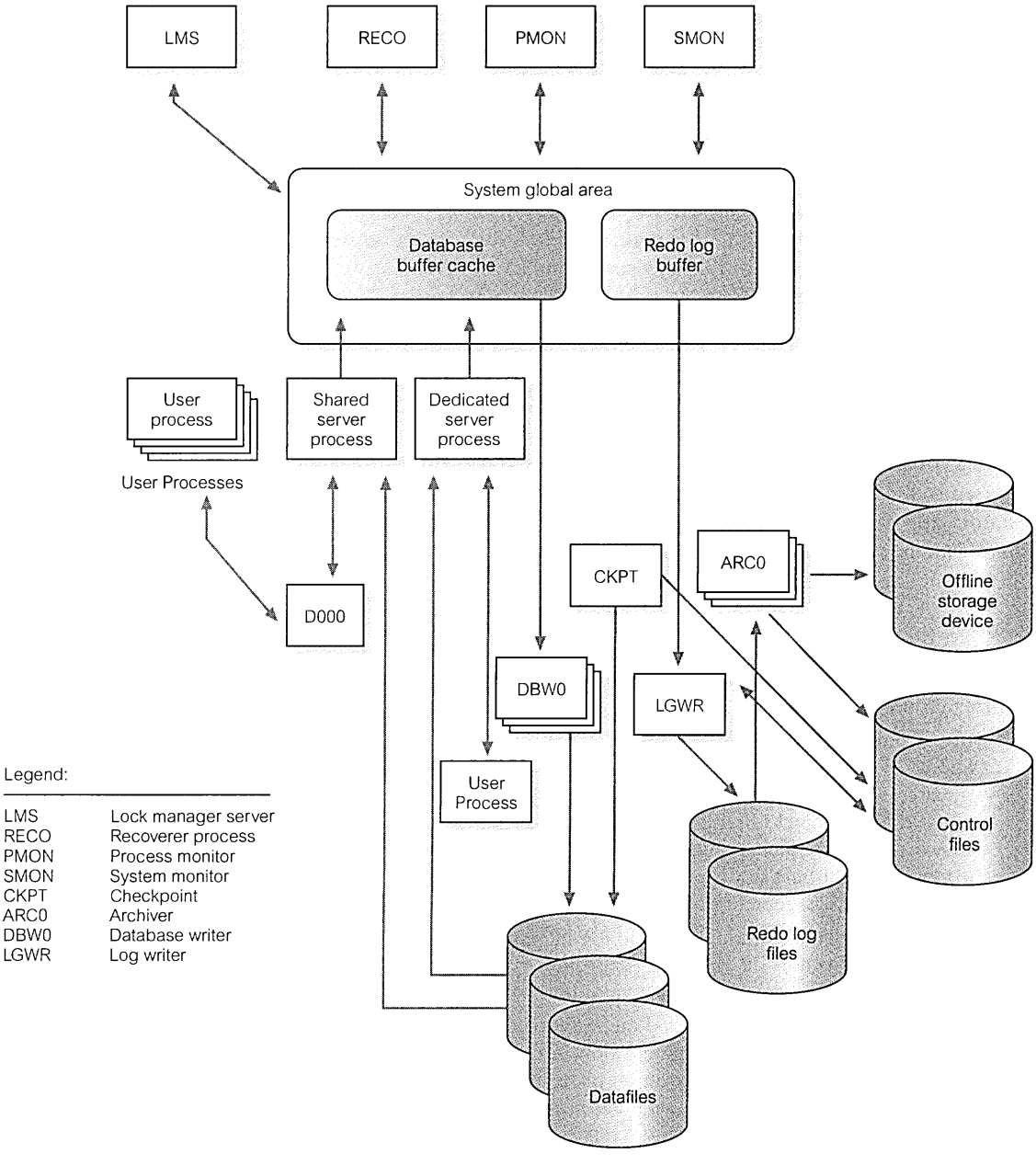

Figure 3.18 The Oracle architecture (from the Oracle documentation set).

- *Oracle processes.* Oracle (server) processes perform functions for users. Oracle processes can be split into two groups: *server processes* (which handle requests from connected user processes) and *background processes* (which perform asynchronous I/O and provide increased parallelism for improved performance and reliability). From Figure 3.18, we have the following background processes:

- *Database Writer (DBWR).* The DBWR process is responsible for writing the modified (dirty) blocks from the buffer cache in the SGA to datafiles on disk. An Oracle instance can have up to ten DBWR processes, named DBW0 to DBW9, to handle I/O to multiple datafiles. Oracle employs a technique known as write-ahead logging (see Section 22.3.4), which means that the DBWR process performs *batched writes* whenever the buffers need to be freed, not necessarily at the point the transaction commits.
- *Log Writer (LGWR).* The LGWR process is responsible for writing data from the log buffer to the redo log.
- *Checkpoint (CKPT).* A checkpoint is an event in which all modified database buffers are written to the datafiles by the DBWR (see Section 22.3.2). The CKPT process is responsible for telling the DBWR process to perform a checkpoint and to update all the datafiles and control files for the database to indicate the most recent checkpoint. The CKPT process is optional and, if omitted, these responsibilities are assumed by the LGWR process.
- *System Monitor (SMON).* The SMON process is responsible for crash recovery when the instance is started following a failure. This includes recovering transactions that have died because of a system crash. SMON also defragments the database by merging free extents within the datafiles.
- *Process Monitor (PMON).* The PMON process is responsible for tracking user processes that access the database and recovering them following a crash. This includes cleaning up any resources left behind (such as memory) and releasing any locks held by the failed process.
- *Archiver (ARCH).* The ARCH process is responsible for copying the online redo log files to archival storage when they become full. The system can be configured to run up to 10 ARCH processes, named ARC0 to ARC9. The additional archive processes are started by the LWGR when the load dictates.
- *Recoverer (RECO).* The RECO process is responsible for cleaning up failed or suspended distributed transactions (see Section 25.4).
- *Dispatchers (Dnnn).* The D*nnn* processes are responsible for routing requests from the user processes to available shared server processes and back again. Dispatchers are present only when the Shared Server (previously known as the Multi-Threaded Server, or MTS) option is used, in which case there is at least one D*nnn* process for every communications protocol in use.
- *Lock Manager Server (LMS).* The LMS process is responsible for interinstance locking when the Oracle Real Application Clusters option is used.

In the previous descriptions we have used the term "process" generically. Nowadays, some systems implement processes as *threads*.

Example of how these processes interact

The following example illustrates an Oracle configuration with the server process running on one machine and a user process connecting to the server from a separate machine. Oracle uses a communication mechanism called Oracle Net Services to allow processes on different physical machines to communicate with each other. Oracle Net Services supports a variety of network protocols, such as TCP/IP. The services can also perform network protocol interchanges, allowing

clients that use one protocol to interact with a database server using another protocol.

(1) The client workstation runs an application in a user process. The client application attempts to establish a connection to the server using the Oracle Net Services driver.

(2) The server detects the connection request from the application and creates a (dedicated) server process on behalf of the user process.

(3) The user executes an SQL statement to change a row of a table and commits the transaction.

(4) The server process receives the statement and checks the shared pool for any shared SQL area that contains an identical SQL statement. If a shared SQL area is found, the server process checks the user's access privileges to the requested data and the previously existing shared SQL area is used to process the statement; if not, a new shared SQL area is allocated for the statement so that it can be parsed and processed.

(5) The server process retrieves any necessary data values from the actual datafile (table) or those stored in the SGA.

(6) The server process modifies data in the SGA. The DBWR process writes modified blocks permanently to disk when doing so is efficient. Because the transaction committed, the LGWR process immediately records the transaction in the online redo log file.

(7) The server process sends a success/failure message across the network to the application.

(8) During this time, the other background processes run, watching for conditions that require intervention. In addition, the Oracle server manages other users' transactions and prevents contention between transactions that request the same data.

Chapter Summary

- **Client–server** architecture refers to the way in which software components interact. There is a **client** process that requires some resource, and a **server** that provides the resource. In the two-tier model, the client handles the user interface and business processing logic and the server handles the database functionality. In the Web environment, the traditional two-tier model has been replaced by a three-tier model, consisting of a user interface layer (the **client**), a business logic and data processing layer (the **application server**), and a DBMS (the **database server**), distributed over different machines.
- The three-tier architecture can be extended to *n* tiers, will additional tiers added to provide more flexibility and scalability.
- **Middleware** is computer software that connects software components or applications. Middleware types include RPC (synchronous and asynchronous), publish/subscribe, message-oriented middleware (MOM), object-request brokers (ORB), and database middleware.
- A **Web service** is a software system designed to support interoperable machine-to-machine interaction over a network. They are based on standards such as XML, SOAP, WSDL, and UDDI.

- **Service-Oriented Architecture** (**SOA**) is a business-centric software architecture for building applications that implement business processes as sets of services published at a granularity relevant to the service consumer.
- A **Transaction Processing** (**TP**) **Monitor** is a program that controls data transfer between clients and servers in order to provide a consistent environment, particularly for online transaction processing (OLTP). The advantages include transaction routing, distributed transactions, load balancing, funneling, and increased reliability.

Review Questions

3.1 What is meant by the term 'client–server architecture' and what are the advantages of this approach? Compare the client–server architecture with two other architectures.

3.2 Compare and contrast the two-tier client–server architecture for traditional DBMSs with the three-tier client–server architecture. Why is the latter architecture more appropriate for the Web?

3.3 What is an *n*-tier architecture?

3.4 What is middleware? Provide a classification service for middleware.

3.5 What is a TP Monitor? What advantages does a TP Monitor bring to an OLTP environment?

3.6 What is a Web service?

3.7 What technologies and standards are used to develop Web services and how do they relate to each other?

3.8 What is a service-oriented architecture?

3.9 Provide an example of a service-oriented architecture.

3.10 Describe the main components in a DBMS.

3.11 Describe the internal architecture of Oracle.

Exercises

3.12 Examine the documentation sets of Microsoft SQL Server, Oracle, and IBM's DB2 system to identify their support for the following:
(a) client–server architecture
(b) Web services
(c) service-oriented architecture

3.13 Search the Web for a number of Web services other than the ones discussed in Section 3.2. What do these services have in common? Identify whether the services access a database.