CHAPTER 2

Database Planning and Database Architecture

Chapter Objectives

In this chapter you will learn the following:

- The distinction between data and information
- The four levels of discussion about data
- The steps in staged database design
- The functions of a database administrator
- The distinction between DDL and DML
- The rationale for and contents of the three-level database architecture
- The meaning of logical and physical data independence

- Characteristics of various data models

On the Companion Website:

- Lab Exercises
- Sample Project
- Student Projects

2.1 Data as a Resource

If you were asked to identify the resources of a typical business organization, you would probably include capital equipment, financial assets, and personnel, but you might not think of data as a resource. When a corporate database exists, the data it contains is a genuine corporate resource. Since the database contains data about the organization's operations (called **operational data**) that is used by many departments, and since it is professionally managed by a DBA, there is an increased appreciation of the value of the data itself, independent of the applications that use it. A **resource** is any asset that is of value to an organization and that incurs costs. An organization's operational data clearly fits this definition. To appreciate the value of an organization's data more fully, imagine what would happen if the data were lost or fell into the hands of a competitor. Many organizations, such as banks and brokerage houses, are heavily dependent on data, and would fail very quickly if their data were lost. Most businesses would suffer heavy losses if their operational data were unavailable. In fact, an organization depends on the availability of operational data in managing its other resources. For example, decisions about the purchase, lease, or use of equipment; financial investments and financial returns; and staffing needs should be made on the basis of information about the organization's operations. The recognition of data as a corporate resource is an important objective in developing an integrated database environment. The database protects the data resource by providing data security, integrity, and reliability controls through the DBMS.

2.2 Characteristics of Data

In order to appreciate the importance of data as a corporate resource, we need to examine its characteristics more closely.

2.2.1 Data and Information

We often think of data as information, but these two terms have slightly different meanings. The term **data** refers to the bare facts recorded in the database. **Information** is processed data that is in a form that is useful for making decisions. Information is derived from the stored data by re-arranging, selecting, combining, summarizing, or performing other operations on the data. For example, if we simply print out all the stored items in the database shown in Figure 1.1, without the column headings that identify what they represent, we have data. However, if we print a formatted report such as the one in Figure 1.3, showing the data in some order that helps us make a decision, we have information. In practice, most people use the two terms interchangeably.

2.2.2 Levels of Discussing Data

When we discuss data, it is important to distinguish between the real world, the small part of the real world that the database is concerned with, the inherent structure of that portion of the world, the structure of the database, and the data stored in the database. There are actually four levels of discussion or abstraction to be considered when we talk about databases.

We begin with the **real world**, or reality. On this first level, we talk about the **enterprise**, the organization for which the database is designed. The enterprise might be a corporation, government agency, university, bank, brokerage house, school, hospital, or other organization. As the enterprise functions in the real world, there is too much detail involved to keep track of all the facts needed for decision making by direct observation. Instead, we identify those facets of the enterprise that we need to keep track of for decision making. The part of the real world that will be represented in the database is called a **miniworld** or a **universe of discourse**. For the miniworld, we begin to develop a **conceptual model**, which forms the second level of data discussion. We identify **entities** which are persons, places, events, objects, or concepts about which we collect data. For the organizations mentioned previously, we could choose entities such as customers,

employees, students, bank accounts, investments, classes, or patients. We group similar entities into **entity sets**. For example, for the set of all customers we form the entity set we might call Customers. Similarly, we might have entity sets called Employees, Students, Accounts, Investments, Classes, and Patients—each consisting of all entity instances of the corresponding type. A conceptual model may have many entity sets. Each entity has certain **attributes**, which are characteristics or properties to describe the entity and that the organization considers important. For the Student entity set, attributes might include student ID, name, address, telephone number, major, credits passed, grade point average, and adviser. Some entities may have **relationships** or associations with other entities. For example, in a university, students are related to classes by being enrolled in those classes, and faculty members are related to classes by teaching them. The conceptual model will represent these relationships by connecting the entity sets in some manner. The concepts of entity, attribute, and relationship will be discussed in more detail in Chapter 3. The conceptual model is often represented by a diagram such as an ER or UML diagram. The model should be designed to be a useful picture of the organization and its operations for the miniworld of interest.

The structure of the database, called the **logical model** of the database, is the third level of discussion. The logical model is also called the **intension** of the database, and it is relatively permanent. The **database schema** is a written description of the logical model. The schema may change occasionally if new data needs arise, a process called **schema evolution**. On this level, we talk about **metadata**, or data about data. For each entity set in the conceptual model, we have a **record type** (or equivalent representation, such as a class) in the logical model of the database. For example, for the Student entity set in the university, we would have a `Student` record type. A record type contains several **data item types**, each of which represents an attribute of an entity. For the `Student` record type, the data item types could be `stuId`, `stuName`, `address`, `phone`, `major`, `credits`, `gpa`, and `adviser`. A **data item** is the smallest named unit of stored data. Other words sometimes used for data item are data element, field, or attribute. Data items are sometimes grouped together to form **data aggregates**, which are named groups of data items within a record. Data aggregates allow us to refer to the group of data items as a whole or to the individual items in the group. A **record** is a named collection of related data items

and/or data aggregates. Relationships must also be represented in the logical model.

Information about the logical structure of the database is stored in a DBMS **data dictionary**, also called a **data directory** or **system catalog**. This repository of information contains descriptions of the record types, data item types, and data aggregates in the database, as well as other information. The data dictionary is actually a database about the database. However, a DBMS system catalog usually does much more than simply store the description of the database structure. For some systems, the system catalog is actively involved in all database accesses.

The fourth level of discussion concerns actual data in the database itself. It consists of **data instances** or occurrences. For each entity occurrence in the miniworld, there will be an occurrence of a corresponding record in the database. For each student in the university, there is a student record occurrence. So, while there is only one `Student` record type, which is described in the data dictionary and corresponds to the Student entity set, there may be thousands of `Student` record occurrences, corresponding to individual student entities, in the database itself. Similarly, there will be many instances of each of the data item types that correspond to attributes. A **file** (sometimes called a data set) is a named collection of record occurrences. For example, the `Student` file may contain 5000 `Student` records. Finally, the **database** may be thought of as a named collection of related files. The data stored in the database at a given moment is called an **extension** of the database, or a **database instance** or **database state**. The extension changes whenever records are added, deleted, or updated. The extension should always be a **valid state**, which means it should satisfy all the constraints specified in the schema. The intension of the database is actually a complex abstract data structure that formally defines all possible extensions. Figure 2.1 summarizes the four levels of discussion of data.

2.2.3 Data Sublanguages

The language that is used to describe a database to a DBMS is part of a **data sublanguage**. A data sublanguage consists of two parts: a **data definition language** (DDL) and a **data manipulation language** (DML). The DDL is used to describe the database, while the DML is used to process the database. These languages are called sublanguages because they have limited functionality, lacking many features of general purpose programming

Realm	Objects	Examples
Real World Containing Miniworld	Enterprise	Corporation, university, bank
	Some aspects of the enterprise	Human resources Student enrollment Customers and accounts
Conceptual Model	Entity	a student, a class
	Attribute	name, schedule
	Entity set	all students, all classes
	Relationship	Student entity relates to Class entity by being enrolled in it
Logical Model Metadata: data definitions, stored in Data Dictionary	Record type	Student record type, Class record type
	Data item type	stuId, classNumber
	Data aggregate	address, consisting of street, city, state, ZIP
Data Ocurrences stored in the database	Student record occurrence	Record of student Tom Smith
	Data item occurrence	'S1001','Smith','Tom','History',90
	File	Student file with 5000 Student records
	Database	University database containing Student file, Class file, Faculty file, . . .

FIGURE 2.1

Four Levels of Discussing Data

languages. The data sublanguage commands can be embedded within a **host** program, a general purpose language program in, for example, C, C++, C#, Java, COBOL, Fortran, or Ada. There are standards for most of the widely used general purpose languages as host languages for standard data sublanguages. In addition, most data sublanguages allow non-embedded or interactive commands for access from workstations.

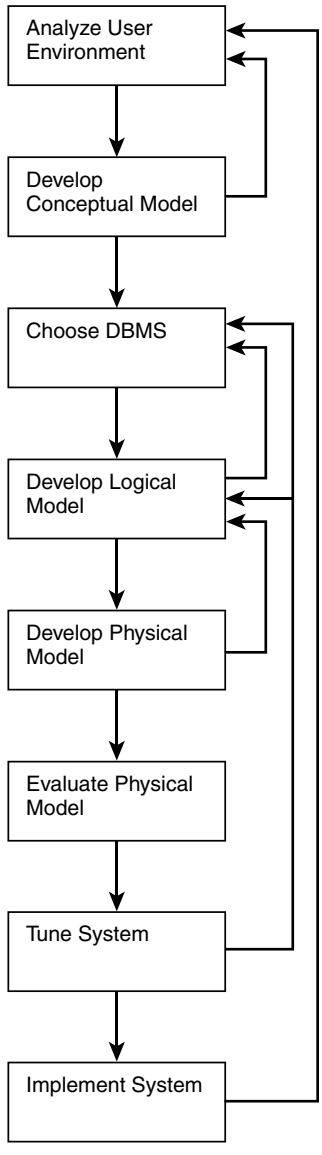
2.3 Stages in Database Design

The process of analyzing the organization and its environment, developing a database model, and implementing that model requires an appropriate methodology. Traditional systems analysis provides a possible approach, but a staged database design approach offers a better solution. The database should be designed in such a way that it can evolve, changing to meet the future information needs of the organization. This evolution is possible when the designer develops a true conceptual model of the organization with the following characteristics:

- The model faithfully mirrors the operations of the organization.
- It is flexible enough to allow changes as new information needs arise.
- It supports many different user views.
- It is independent of physical implementation.
- It does not depend on the data model used by a particular database management system.

A well-designed conceptual database model protects the data resource by allowing it to evolve so that it serves both today's and tomorrow's information needs. Even if the database management system chosen for implementation is replaced, the logical model may change, but the conceptual model of the enterprise can survive. The staged database design approach is a top-down method that begins with general statements of needs, and progresses to more and more detailed consideration of problems. Different problems are considered at different phases of the project. Each stage

FIGURE 2.2
Steps in Staged Database Design



uses design tools that are appropriate to the problem at that level. Figure 2.2 shows the major design stages. They are

1. Analyze the user environment.
   The first step in designing a database is to determine the current user environment. The designer studies all current applications,

determines their input and output, examines all reports generated by the present system, and interviews users to determine how they use the system. After the present system is thoroughly understood, the designer works closely with present users and potential users of the new system to identify their needs using these same methods. The designer considers not only present needs but possible new applications or future uses of the database. The result of this analysis is a model of the user environment and requirements.

2. Develop a conceptual data model.
   Using the model of the user environment, the designer develops a detailed conceptual model of the database—identifying the entities, attributes, and relationships that will be represented. In addition to the conceptual model, the designer has to consider how the database is to be used. The types of applications and transactions, the kinds of access, the volume of transactions, the volume of data, the frequency of access, and other quantitative data must be specified. Other constraints such as budgetary restrictions and performance requirements must also be identified. The result of this phase is a set of database specifications.

3. Choose a DBMS.
   The designer uses the specifications and knowledge of available hardware and software resources to evaluate alternative database management systems and chooses the system that best satisfies the specifications.

4. Develop the logical model.
   The designer maps the conceptual model to the data model used by the chosen DBMS, creating the logical model.

5. Develop the physical model.
   The designer plans the layout of data considering the structures supported by the chosen DBMS, and hardware and software resources available.

6. Evaluate the physical model.
   The designer estimates the performance of all applications and transactions, considering the quantitative data previously identified, information about the performance characteristics of hardware and software to be used, and priorities given to applications

and transactions. It can be helpful to develop a **prototype**, implementing a selected portion of the database so that user views can be validated and performance measured more accurately.

7. Perform tuning if indicated by the evaluation.
   Adjustments such as modifying physical structures or optimizing software can be done to improve performance.
8. Implement the physical model.
   If the evaluation is positive, the designer then implements the physical design and the database becomes operational.

The loops in Figure 2.2 provide opportunities for feedback and changes at various stages in the design process. For example, after developing the conceptual model, the designer communicates with user groups to make sure their data requirements are represented properly. If not, the conceptual model must be adjusted. If the conceptual model does not map well to a particular data model, another should be considered. For a particular DBMS, there might be several possible logical mappings that can be evaluated. If the physical model is not acceptable, a different mapping or a different DBMS can be considered. If the results of the performance evaluation are not satisfactory, additional tuning and evaluation may be performed. The physical model can be changed if tuning does not produce the required performance. If repeated tuning and optimization are not sufficient, it might be necessary to change the way the logical model is mapped to the DBMS or to consider a different database management system. Even after the system is implemented, changes might be needed to respond to changing user needs or a changing environment.

2.4 Design Tools

Many methodologies exist that can make the database design process easier for both designers and users. They vary from general techniques described in the software engineering literature to commercial products to automate the design process. For example, **CASE** (Computer-Aided Software Engineering) packages include various tools for system analysis, project management, and design that can be very useful in the database design process. They are helpful for collecting and analyzing data, designing the data model, designing applications, and implementing

the database, including prototyping, data conversion, generating application code, generating reports, and testing. **Project management software** is another type of tool that can be applied effectively to database development.

2.4.1 Data Dictionary

A DBMS data dictionary, as discussed in Section 2.2.2, is a repository of information that describes the logical structure of the database. It contains **metadata**, or data about the data in the database. It has entries for all the types of objects that exist in the database. If the data dictionary is part of the DBMS, it is referred to as an **integrated** data dictionary or **system catalog**. A system catalog is always consistent with the actual database structure, because it is maintained automatically by the system. It can perform many functions throughout the life of the database. If the data dictionary is available without a particular DBMS, we refer to it as a **freestanding** data dictionary. A freestanding data dictionary can be a commercial product or a simple file developed and maintained by the designer. For example, CASE packages often include a data dictionary tool, and freestanding data dictionaries are available even for non-database environments. Both integrated and freestanding data dictionaries have advantages and disadvantages, but a freestanding dictionary is helpful in the initial design stages, before the designer has chosen a particular DBMS. It allows the designer to enjoy the advantages of having this tool without committing to a particular implementation. A major disadvantage is that once the database is created, adjustments to its structure may not be entered into the freestanding data dictionary and, over time, the dictionary will not be an accurate reflection of the database structure.

A freestanding data dictionary is useful in the early design stages for collecting and organizing information about data. Each data item, its source(s), its name(s), its uses, its meaning, its relationship to other items, its format, and the identity of the person or group responsible for entering it, updating it, and maintaining its correctness must be determined. The data dictionary provides an effective tool for accomplishing these tasks. The database administrator (DBA) can begin development of the data dictionary by identifying data items, securing agreement from users on a definition of each item, and entering the item in the dictionary. Since users

should not be aware of other users' data, the DBA should control access to the dictionary.

A freestanding data dictionary is useful for the following:

- Collecting and storing information about data in a central location. This helps management to gain control over data as a resource.
- Securing agreement from users and designers about the meanings of data items. An exact, agreed-upon definition of each item should be developed for storage in the data dictionary.
- Communicating with users. The data dictionary greatly facilitates communication, since exact meanings are stored. The dictionary also identifies the persons, departments, or groups having access to or interest in each item.
- Identifying redundancy and inconsistency in data item names. In the process of identifying and defining data items, the DBA may discover **synonyms**, which are different names for the same item. The database can accept synonyms, but the system must be aware of them. The DBA might also discover **homonyms**, which are identical names for different data items. These are never permitted in a database.
- Keeping track of changes to the database structure. Changes such as creation of new data items and record types or alterations to data item descriptions should be recorded in the data dictionary.
- Determining the impact of changes to the database structure. Since the data dictionary records each item, all its relationships, and all its users, the DBA can see what effects a change would have.
- Identifying the sources of and responsibility for the correctness of each item. A general rule for data entry is that data should be captured as near to its source as possible. The persons or departments that generate or capture values for each data item and those responsible for updating each item should be listed in the data dictionary.

- Recording the external, logical, and internal schemas and the mappings between them. This aspect will be discussed in Section 2.6.
- Recording access control information. A data dictionary can record the identities of all those who are permitted to access each item, along with the type of access-retrieval, insertion, update, or deletion.
- A DBMS system catalog can also perform all these functions, but it is also useful for providing audit information. The system catalog can be used to record each access, allowing usage statistics to be collected for auditing the system and spotting attempted security violations.

Note that not all data dictionaries provide support for all of these functions, and some provide additional functions. Some just store the schema, some control documentation, while some system catalogs are "meta-systems" that control access to the database, generate system code, and keep statistics on database usage as well.

2.4.2 Project Management Software

This type of software provides a set of tools that can be used to plan and manage a project, especially when there are many people working on it. There are usually several types of graphs and charts available, such as **Gantt charts** and **PERT charts**, which are similar. The user specifies the objectives and scope of the project, identifies the major tasks and phases, indicates dependencies among the tasks, identifies the resources available, and sets a timeline for completion of the tasks and phases of the project. The software can be used to generate calendars, to produce graphs with many different views of the progress of the project, and to provide a means of communication among the project staff, using either intranet or Internet access. An example is Microsoft Project.

2.5 Database Administration

The database administrator is responsible for the design, operation, and management of the database. In many cases, the conceptual design is done by a database designer, and the DBA implements the design, develops the

system, and manages it. The DBA must be technically competent, a good manager, a skilled communicator, and must have excellent interpersonal and communication skills. The DBA has many functions that vary according to the stage of the database project. Major functions include planning, designing, developing, and managing the database.

2.5.1 Planning and Design

- **Preliminary Database Planning.** If the DBA or database designer is chosen early enough in the project, he or she should participate in the preliminary investigation and feasibility study.
- **Identifying User Requirements.** The DBA or designer examines all transactions and all reports generated by the present system and consults with users to determine whether they satisfy their information needs. He or she works with present and potential users to design new reports and new transactions. The frequency of reports and transactions and the timeframe within which they must be produced are also recorded. The DBA uses his or her knowledge of the long-term and short-term objectives of the organization to prioritize user requirements.
- **Developing and Maintaining the Data Dictionary.** As he or she determines data needs of users, the DBA or designer stores data item names, sources, meanings, usage, and synonyms in the data dictionary. The DBA revises the data dictionary as the project progresses.
- **Designing the Conceptual Model.** The DBA or designer identifies all entities, attributes, and relationships that are to be represented in the database, and develops a conceptual model that is an accurate reflection of the miniworld.
- **Choosing a DBMS.** The DBA considers the conceptual model and other database specifications and the hardware and software available, and chooses the DBMS that best fits the environment and meets the specifications.
- **Developing the Logical Model.** There may be several ways the conceptual model could be mapped to the data model used by the DBMS. The DBA chooses the one that appears to be most appropriate.

- **Developing the Physical Model.** There may be several ways the logical model could be mapped to the data structures provided by the DBMS and to physical devices. The DBA evaluates each mapping by estimating performance of applications and transactions. The best mapping becomes the physical model.

2.5.2 Developing the Database

- **Creating and Loading the Database.** Once the physical model is developed, the DBA creates the structure of the database using the data definition language for the chosen DBMS. He or she establishes physical data sets, creates libraries, and loads the data into the database.
- **Developing User Views.** The DBA attempts to satisfy the data needs of all users. A user's view may be identical to the one requested in the initial design stages. Often, however, users' requests change as they begin to understand the system better. If the view does not coincide with the user's request, the DBA should present cogent reasons why the request has not been met and secure agreement on the actual view.
- **Writing and Maintaining Documentation.** When the database is created, the DBA makes sure all documentation accurately reflects the structure of the database. Some database documentation is written automatically by the system catalog as the database is created.
- **Developing and Enforcing Data Standards.** Since the database is shared by many users, it is important that standards be defined and enforced for the benefit of all. Users who are responsible for inserting and updating data must follow a standard format for entering data. The user interface should be designed to make it easy for users to follow the standards by displaying default values, showing acceptable ranges for items, and so on. Typical data standards include specifications for null values, abbreviations, codes, punctuation, and capitalization. The system can automatically check for errors and range restrictions, uniqueness of key values, and relationships between data values in a single record, between records in the same file, and between records in different files.
- **Developing and Enforcing Application Program Standards.** The DBA must develop security and privacy standards for applications

and make sure they are subject to the audit mechanism, make proper use of the high-level data manipulation language, and fit the application development facilities provided by the DBMS. These standards apply both to old applications that are converted for use with the database and to new applications.

- **Developing Operating Procedures.** The DBA is responsible for establishing procedures for daily startup of the DBMS (if necessary), smooth running of database operations, logging of transactions, periodic backups, security and authorization procedures, recording hardware and software failures, taking performance measurements, shutting down the database in an orderly fashion in case of failure, restarting and recovering after failure, and shutting down at the end of each day (if necessary). Since these procedures are performed by operators, the DBA must consult with the operations manager to ensure that operators are trained in all aspects of database operations.
- **Doing User Training.** End users, application programmers, and systems programmers who access the database should participate in training programs so that they can learn to use it most effectively. Sessions may be conducted by the DBA, the vendor of the DBMS, or other technical trainers, either onsite or in a training center.

2.5.3 Database Management

- **Monitoring Performance.** The DBA is responsible for collecting and analyzing statistics on database performance and responding to user complaints and suggestions regarding performance. The running time for applications and the response time for interactive queries should be measured, so that the DBA can spot problems in database use. Usually, the DBMS provides facilities for recording this information. The DBA continually compares performance to requirements and makes adjustments when necessary.
- **Tuning and Reorganizing.** If performance begins to degrade as changes are made to the stored data, the DBA can respond by adding or changing indexes, reorganizing files, using faster storage devices, or optimizing software. For serious performance problems, he or she may have to reorganize the entire database.
- **Keeping Current on Database Improvements.** The DBA should be aware of new features and new versions of the DBMS that become

available. He or she should evaluate these new products and other hardware and software developments to determine whether they would provide substantial benefits to the organization.

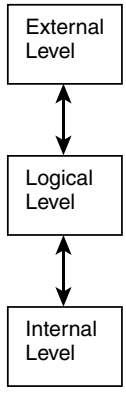
2.6 The Three-Level Database Architecture

In Section 2.3, we presented a staged database design process that began with the development of a conceptual model and ended with a physical model. Now we are ready to examine these and related concepts more closely. When we discuss a database, we need some means of describing different aspects of its structure. An early proposal for a standardized vocabulary and architecture for database systems was developed and published in 1971 by the Database Task Group appointed by the Conference on Data Systems and Languages, or CODASYL DBTG. As a result of this and later reports, databases can be viewed at three levels of abstraction. The levels form a layered **three-level architecture** and are described by three **schemas**, which are written descriptions of their structures. The purpose of the three-level architecture is to separate the user's model from the physical structure of the database. There are several reasons why this separation is desirable:

- Different users need different views of the same data.
- The way a particular user needs to see the data may change over time.
- Users should not have to deal with the complexities of the database storage structures.
- The DBA should be able to make changes to the conceptual model of the database without affecting all users.
- The DBA should be able to change the logical model, including data structures and file structures, without affecting the conceptual model or users' views.
- Database structure should be unaffected by changes to the physical aspects of storage, such as changes to storage devices.

Figure 2.3 shows a simplified version of the three-level architecture of database systems. The way users think about data is called the **external level**. The **internal level** is the way the data is actually stored using standard data structures and file organizations. However, there are many different users' views and many physical structures, so there must be some method of mapping the external views to the physical structures. A direct mapping is undesirable, since changes made to physical structures or storage devices would require a

FIGURE 2.3
Simplified Three-Level Architecture for Database Systems



corresponding change in the external to physical mapping. Therefore there is a middle level that provides both the mapping and the desired independence between the external and physical levels. This is the **logical** level.

2.6.1 External Views

The **external level** consists of many different **external views** or **external models** of the database. Each user has a model of the real world represented in a form that is suitable for that user. Figure 2.4 presents a more detailed picture of the three-level database architecture. The top level of Figure 2.4 shows several external views. A particular user interacts with only certain aspects of the miniworld, and is interested in only some entities, and only some of their attributes and relationships. Therefore, that user's view will contain only information about those aspects. Other entities or other attributes or relationships may actually be represented in the database, but the user will be unaware of them. Besides including different entities, attributes, and relationships, different views may have different representations of the same data. For example, one user may believe dates are stored in the form month, day, year, while another may believe they are represented as year, month, day. Some views might include **virtual** or calculated data, data not actually stored as such, but created when needed. For example, age may not be actually stored, but dateOfBirth may be, and age may be calculated by the system when the user refers to it. Views may even include data combined or calculated from several records. An **external record** is a record as seen by a particular user, a part of his or her external view. An external view is actually a collection of external records. The external views are described in **external schemas** (also called **subschemas**) that are written in the data definition language (DDL). Each

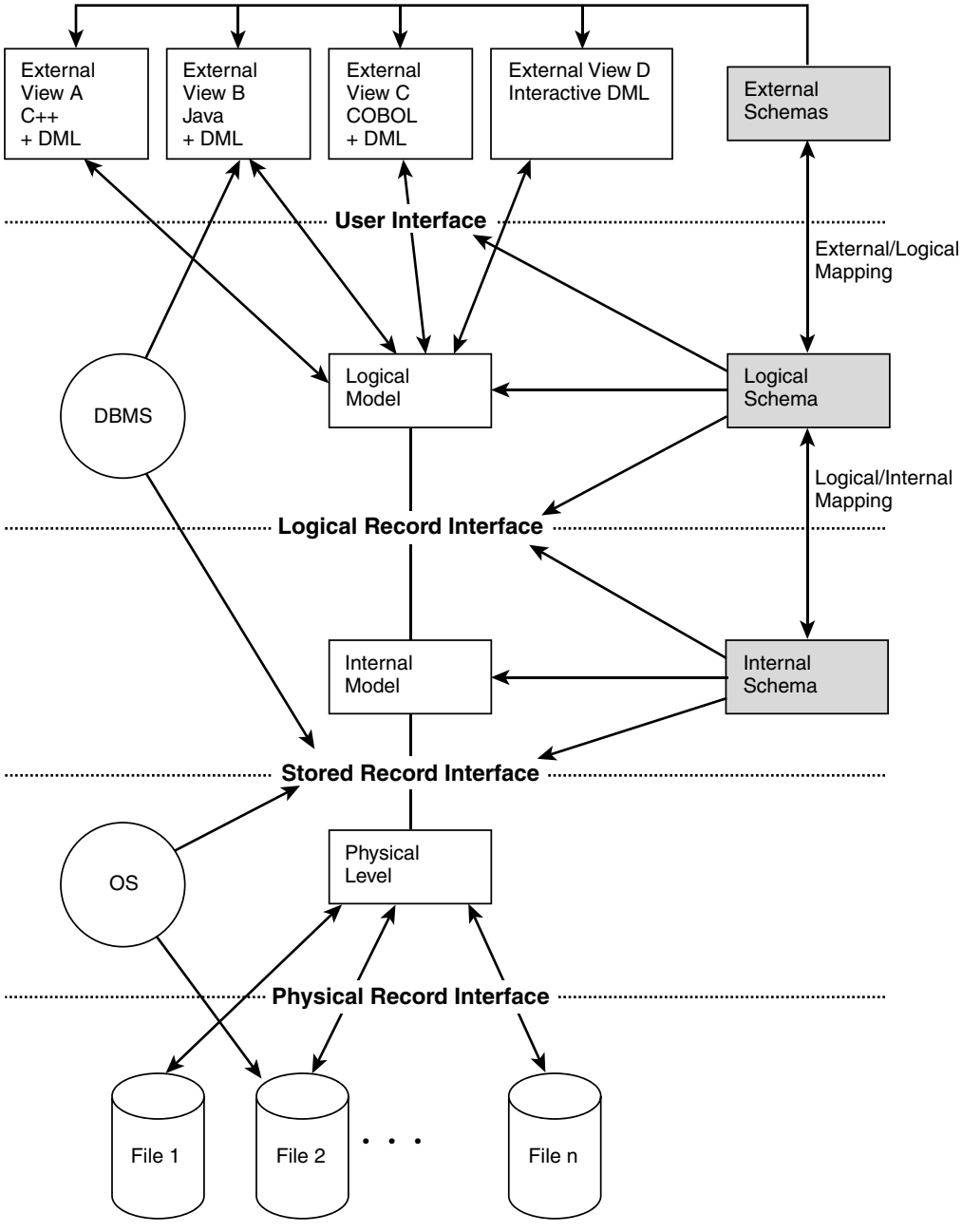


FIGURE 2.4

Three-Level Database Architecture

user's schema gives a complete description of each type of external record that appears in that user's view. The schemas are stored for use by the system data catalog in retrieving records. They should also be kept in source form as documentation. The DBMS uses the external schema to create a **user interface**, which is both a facility and a barrier. An individual user sees the database through this interface. It defines and creates the working environment for that user, accepting and displaying information in the format the user expects. It also acts as a boundary below which the user is not permitted to see. It hides the logical, internal, and physical details from the user.

2.6.2 Logical and Conceptual Models

The middle level in the three-level architecture is the **logical level**, as shown in Figure 2.4. This model includes the entire information structure of the database, as seen by the DBA. It is the "community view" of data, and includes a description of all of the data that is available to be shared. It is a comprehensive model or view of the workings of the organization in the miniworld. All the entities, with their attributes and relationships, are represented in the logical model using the data model that the DBMS supports. The model includes any constraints on the data and semantic information about the data meanings. The logical model supports the external views, in that any data available to any user must be present in or derivable from the logical model. The logical model is relatively constant. When the DBA originally designs it, he or she tries to determine present and future information needs and attempts to develop a lasting model of the organization. Therefore, as new data needs arise, the logical model might already contain the objects required. If that is not the case, the DBA expands the logical model to include the new objects. A good logical model will be able to accommodate this change and still support the old external views. Only users who need access to the new data should be affected by the change. The **logical schema** is a complete description of the information content of the database. It is written in DDL, compiled by the DBMS, and stored in the system catalog and in source form as documentation. The DBMS uses the logical schema to create the **logical record interface**, which is a boundary below which everything is invisible to the logical level and which defines and creates the working environment for the logical level. No internal or physical details such as how

records are stored or sequenced cross this boundary. The logical model is actually a collection of logical records.

The logical data model is the heart of the database. It supports all the external views and is, in turn, supported by the internal model. However, the internal model is merely the physical implementation of the logical model. The logical model is itself derived from the conceptual model. Developing the conceptual model is the most challenging, interesting, and rewarding part of database design. The database designer must be able to identify, classify, and structure elements in the design. The process of **abstraction**, which means identifying common properties of a set of objects rather than focusing on details, is used to categorize data. In computer science, abstraction is used to simplify concepts and hide complexity. For example, abstract data types are considered apart from their implementation; the behavior of queues and stacks can be described without considering how they are represented. The designer can look at different levels of abstraction, so that an abstract object on one level becomes a component of a higher level abstraction. During the conceptual design process, there might be many errors and several false starts. Like many other problem-solving processes, conceptual database design is an art, guided by knowledge. There may be many possible solutions, but some are better than others. The process itself is a learning situation. The designer gradually comes to understand the workings of the organization and the meanings of its data, and expresses that understanding in the chosen model. If the designer produces a good conceptual model, it is a relatively easy task to convert it to a logical model and complete the internal and physical design. If the conceptual model is a good one, the external views are easy to define as well. If any data that a user might be interested in is included in the conceptual model, it is an easy task to put it into the user's external view. On the other hand, a poor conceptual model can be hard to implement, particularly if data and relationships are not well defined. It will also be inadequate to provide all the needed external models. It will continue to cause problems during the lifetime of the database, because it will have to be "patched up" whenever different information needs arise. The ability to adjust to change is one of the hallmarks of good conceptual design. Therefore, it is worthwhile to spend all the time and energy necessary to produce the best possible conceptual design. The payoff will be felt not only at the logical and internal design stages but in the future.

2.6.3 Internal Model

The **internal level** covers the physical implementation of the database. It includes the data structures and file organizations used to store data on physical storage devices. The DBMS chosen determines, to a large extent, what structures are available. It works with the operating system access methods to lay out the data on the storage devices, build the indexes, and/or set the pointers that will be used for data retrieval. Therefore, there is actually a physical level below the one the DBMS is responsible for—one that is managed by the operating system under the direction of the DBMS. The line between the DBMS responsibilities and the operating system responsibilities is not clear cut and can vary from system to system. Some DBMSs take advantage of many of the operating system access method facilities, while others ignore all but the most basic I/O managers and create their own alternative file organizations. The DBA must be aware of the possibilities for mapping the logical model to the internal model, and choose a mapping that supports the logical view and provides suitable performance. The **internal schema**, written in DDL, is a complete description of the internal model. It includes such items as how data is represented, how records are sequenced, what indexes exist, what pointers exist, and what hashing scheme, if any, is used. An **internal record** is a single stored record. It is the unit that is passed up to the internal level. The **stored record interface** is the boundary between the physical level, for which the operating system may be responsible, and the internal level, for which the DBMS is responsible. This interface is provided to the DBMS by the operating system. In some cases where the DBMS performs some operating system functions, the DBMS itself may create this interface. The physical level below this interface consists of items only the operating system knows, such as exactly how the sequencing is implemented and whether the fields of internal records are actually stored as contiguous bytes on the disk. The operating system creates the **physical record interface**, which is a lower boundary where storage details, such as exactly what portion of what track contains what data, are hidden.

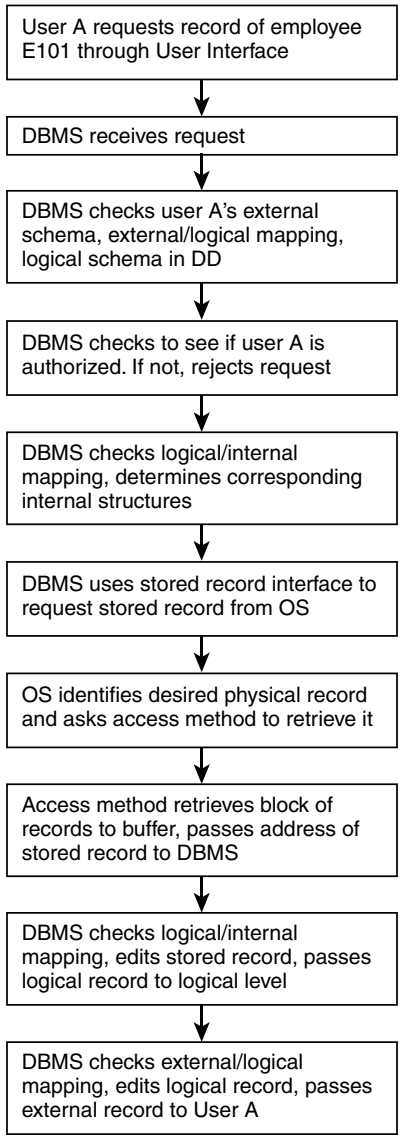
The data dictionary/directory not only stores the complete external, logical, and internal schemas, but it also stores the mappings between them. The **external/logical mapping** tells the DBMS which objects on the logical level correspond to which objects in a particular user's external view. There

may be differences in record names, data item names, data item order, data types, and so forth. If changes are made to either an external view or the logical model, the mappings must be changed. Similarly, the **logical/internal mapping** gives the correspondence between logical objects and internal ones, in that it tells how the logical objects are physically represented. If the stored structure is changed, the mapping must be changed accordingly.

To understand the distinctions among the three levels, we will examine what each level receives and passes on when we request the record of a particular employee. Refer to Figure 2.5 along with Figure 2.6. When User A requests a record such as the (external) record of Employee 101, the DBMS intercepts the request. Once the DBMS receives the request, it checks the user's external view and the external/logical mapping. Both the views and the mapping are stored in object form in the system catalog, so the checking can be accomplished easily. The DBMS also checks the user's authorization to access the items and to perform the operations requested. If there is no such authorization, the request is denied. If the user is authorized, the DBMS notes what logical level objects are needed, and the request is passed down to the logical level. Next, the logical/internal mapping is checked to see what internal structures correspond to the logical items. Once again, the system catalog has stored both of the models and the mapping. The DBMS identifies the internal objects that are required, and it passes the request to the operating system.

On the physical level, an employee record is contained in a physical record, a page or block that can hold several employee records. This is the unit that is brought to the buffer from the disk. The operating system is responsible for performing the basic job of locating the correct block for the requested record and managing its retrieval. When the retrieval is complete, the proper block is in the buffer, and the operating system passes to the DBMS the exact location within the buffer at which the stored record appears. The DBMS reads only the requested employee record, not all the records in the block. However, it sees the complete stored record, exactly as it is coded or encrypted, together with any embedded blanks and pointers that may appear in it, but without its header. This is a description of an internal record, which is the unit that passes up through the stored record interface. The DBMS uses the logical/internal mapping to decide what items to pass up through the logical record interface to the logical level.

FIGURE 2.5
Retrieving Record of E101 for User A



At the logical level, the record appears as a logical record, with encryption and special coding removed. Pointers that are used for establishing relationships do not appear on the logical level, since that level is only concerned with the existence of relationships, not how they are implemented. Similarly, pointers used for sequencing are not of interest on the logical level, nor are indexes. The logical level record therefore contains only the

External Employee Record:

employeeName	empNumber	dept
JACK JONES	E101	Marketing

Logical Employee Record:

empId	lastName	firstName	dept	salary
E101	Jones	Jack	12	55000

Stored Employee Record:

empId	lastName	firstName	dept	salary	forward pointer	backward pointer
E101bbbbbb	**Jones**bbbbbb	**Jack**bbbbbbbbb	**12**bbbbbbbb	**55000**bbbb	**10101**bbbbbb	**10001**

Physical Record:

Block header	rec of E90	rec of E95	**rec of E101**	rec of E125

FIGURE 2.6
Differences in External, Logical, Stored, and Physical Record

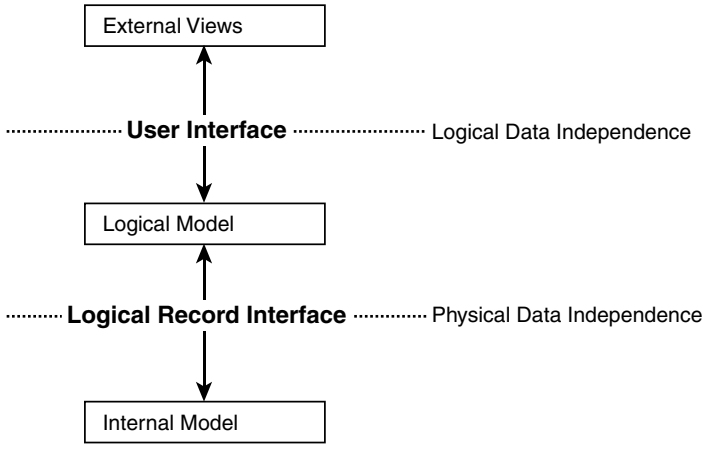
information for that particular employee, but it contains all the fields stored for the employee, in the order in which they are stored. The DBMS checks the external/logical mapping to decide what the user's external record should look like.

When the record is passed up through the user interface to the external level, certain fields can be hidden, some can have their names changed, some can be rearranged, some can appear in a form different from their stored form, and some can be virtual, created from the stored record. Some external records might be combinations of stored records, or the result of operations such as calculations on stored records. The user then performs operations on the external record. That is, he or she can manipulate only those items that appear in the external record. These changes, if legal, are eventually made to the stored record. Figure 2.5 summarizes the steps in this process, and Figure 2.6 illustrates the differences in the appearance of the employee record as it is passed up to the external level.

2.6.4 Data Independence

A major reason for the three-level architecture is to provide **data independence**, which means that upper levels are unaffected by changes to lower

FIGURE 2.7
Logical and Physical Data Independence



levels. There are two kinds of data independence: **logical** and **physical**. Logical data independence refers to the immunity of external models to changes in the logical model. Logical model changes such as addition of new record types, new data items, and new relationships should be possible without affecting existing external views. Of course, the users for whom the changes are made need to be aware of them, but other users should not be. In particular, existing application programs should not have to be rewritten when logical level changes are made.

Physical data independence refers to the immunity of the logical model to changes in the internal model. Internal or physical changes such as a different physical sequencing of records, switching from one access method to another, changing the hashing algorithm, using different data structures, and using new storage devices should have no effect on the logical model. On the external level, the only effect that may be felt is a change in performance. In fact, deterioration in performance is the most common reason for internal model changes. Figure 2.7 shows where each type of data independence occurs.

2.7 Overview of Data Models

A **data model** is a collection of tools often including a type of diagram and specialized vocabulary for describing the structure of the database. There is much disagreement over what constitutes a data model, and that is reflected in the dozens of proposed models and methodologies found in the literature. A data model provides a description of the database structure

including the data, the relationships, constraints, and sometimes data semantics or meanings.

2.7.1 The Entity-Relationship Model

The Entity-Relationship model is an example of a **semantic model**. Semantic models are used to describe the conceptual and external levels of data, and are independent of the internal and physical aspects. In addition to specifying what is to be represented in the database, they attempt to incorporate some meanings or semantic aspects of data such as explicit representation of objects, attributes, and relationships, categorization of objects, abstractions, and explicit data constraints. Some of the concepts of the E-R model were introduced in Section 2.2.2, when we described the four levels of abstraction in the discussion of data. The model was introduced by Chen in the mid 1970s and is widely used for conceptual design. It is based on identifying **entities** that are representations of real objects in the miniworld. Entities are described by their attributes and are connected by relationships. We described entities as persons, places, events, objects, or concepts about which we collect data. A more abstract description is that an entity is any object that exists and is distinguishable from other objects. The **attributes** are qualities or characteristics that describe the entities and distinguish them from one another. We defined an **entity set** as a collection of entities of the same type. Now we also define a **relationship set** as a set of relationships of the same type, and we add the fact that relationships themselves might have **descriptive attributes**. The E-R model also allows us to express constraints, or restrictions, on the entities or relationships. Chapter 3 contains a more complete description of the E-R model, including details about constraints.

One of the most useful and attractive features of the E-R model is that it provides a graphical method for depicting the conceptual structure of the database. E-R diagrams contain symbols for entities, attributes, and relationships. Figure 2.8 shows some of the symbols, along with their names, meanings, and usages. Figure 2.9 illustrates a simple E-R diagram for a students and classes database similar to the one we discussed in Section 1.2. It shows an entity set called Student, with the attributes stuId (the primary key, which is a unique identifier for each student entity instance), lastName, firstName, major, and credits. Class information, to be kept for each class offered during the current term, includes the classNumber (primary key), the schedule, and the room. The Student and Class entity sets are connected by a relationship set, Enroll, which tells us which students

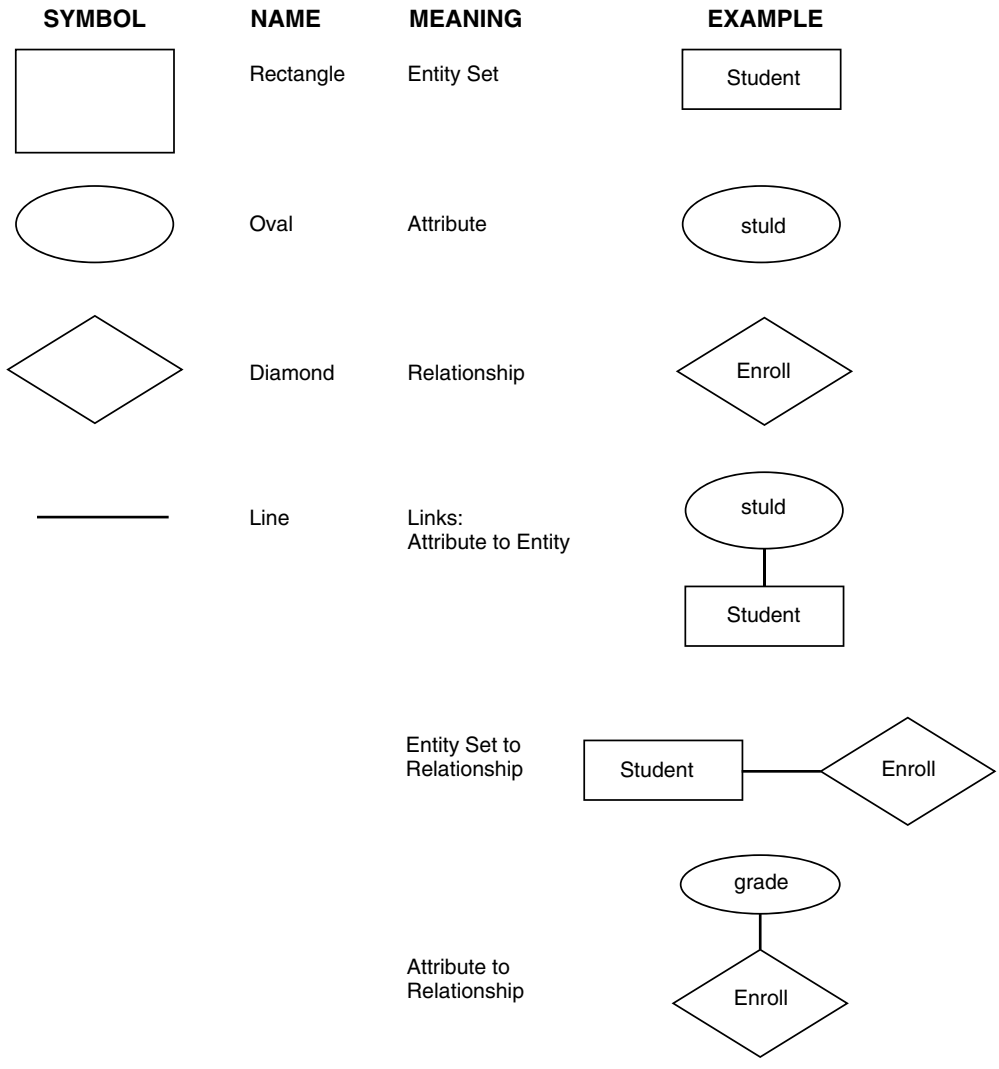


FIGURE 2.8

Basic Symbols for E-R Diagrams

are enrolled in which classes. It has its own descriptive attribute, grade. Note that grade is not an attribute of Student, since knowing the grade for a student is meaningless unless we know the course as well. Similarly, grade is not an attribute of Class, since knowing that a particular grade was given for a class is meaningless unless we know to which student the grade was given. Therefore, since grade has meaning only for a particular combination of student and class, it belongs to the relationship set. Since

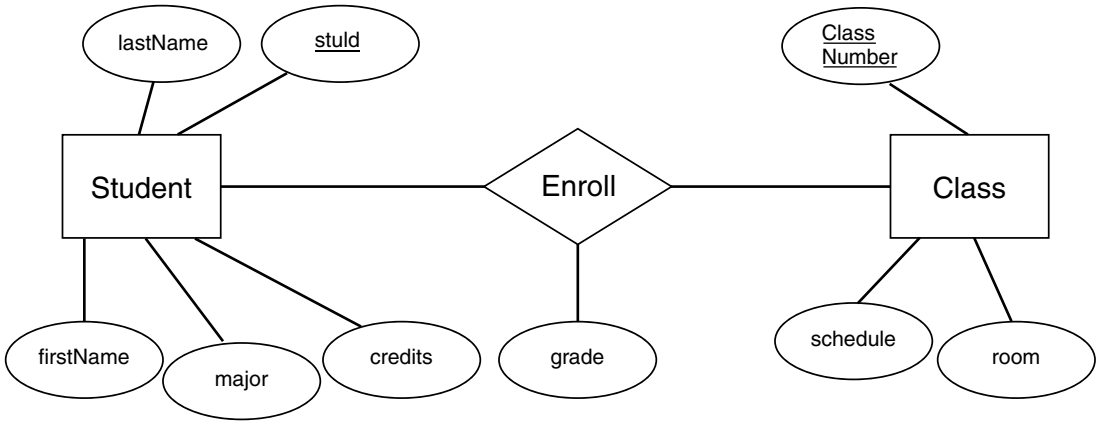


FIGURE 2.9
Simplified E-R Diagram

the E-R model describes only a conceptual structure for the database, we do not attempt to describe how the model could or should be represented internally. Therefore, the material in Section 2.2.2 in which we described data items, records, and files is not part of the E-R model itself.

2.7.2 Relational Model

The relational model is an example of a **record-based model**, one that essentially pictures what the logical records will look like. Record-based models are used to describe the external, logical, and to some extent the internal levels of the database. They allow the designer to develop and specify the logical structure and provide some options for implementation of the design. However, they do not provide much semantic information such as categorization of objects, relationships, abstraction, or data constraints. The relational model was proposed by Codd in 1970 and is widely used, because of its simplicity and its power. The relational model began by using the theory of relations in mathematics and adapting it for use in database theory. The same type of theoretical development of the subject, complete with formal notation, definitions, theorems, and proofs that we usually find in mathematics can be applied to databases using this model. The results of this theoretical development are then applied to practical considerations of implementation. In the relational model, entities are represented as relations, which are physically represented as tables, and attributes as columns of those tables. Some relationships may also be represented as relations or tables. Figure 1.1 (a)–(d) showed a sample relational

database for data about students, faculty, and their classes. We will study the relational model, including SQL, the standard language for the model, in later chapters. Two older record-based models are the **network** and the **hierarchical** models mentioned in Section 1.6. They are primarily of historic interest, since they are no longer widely used for developing new databases. However, many legacy databases based on these models still exist, with code that is still being used and that must be maintained.

2.7.3 Object-Oriented Model

The object-oriented model is a semantic model. It is based on the notion of an **object**, which, like an entity, is a representation of some person, place, event, or concept in the real world that we wish to represent in the database. While entities have only attributes, objects have both a **state** and a **behavior**. The state of an object is determined by the values of its attributes (instance variables). The behavior is the set of functions (methods) defined for the object. A **class** is similar to an entity set and consists of the set of objects having the same structure and behavior. The object-oriented model uses **encapsulation**, incorporating both data and functions in a unit where they are protected from modification from outside. Classes can be grouped into hierarchies, For example, we could define a Person class with attributes such as FirstName, LastName, and ID and methods such as ChangeName. We could then create a Student class and a Faculty class as subclasses of Person. The subclasses inherit the attributes and methods of the parent, so in this example, every Student object and every Faculty object has the attributes and methods of Person. They can also have their own attributes and methods. For example, the Student class can have its own additional attribute, Major, and a method changeMajor. Classes can also participate in relationships of various types. For example, the Student class can be related to the Course class. Unified Modeling Language (UML) class diagrams are usually used for representing object-oriented data models. Figure 2.10 shows an example of a simple OO data model. Every object in a database must have a unique **object identifier** that functions as a permanent primary key, but that does not take its value from any of the object's attributes. An important difference between program objects and database objects is **persistence**. Unlike a program object that exists only while the program is executing, a database object remains in existence after execution of an application program completes.

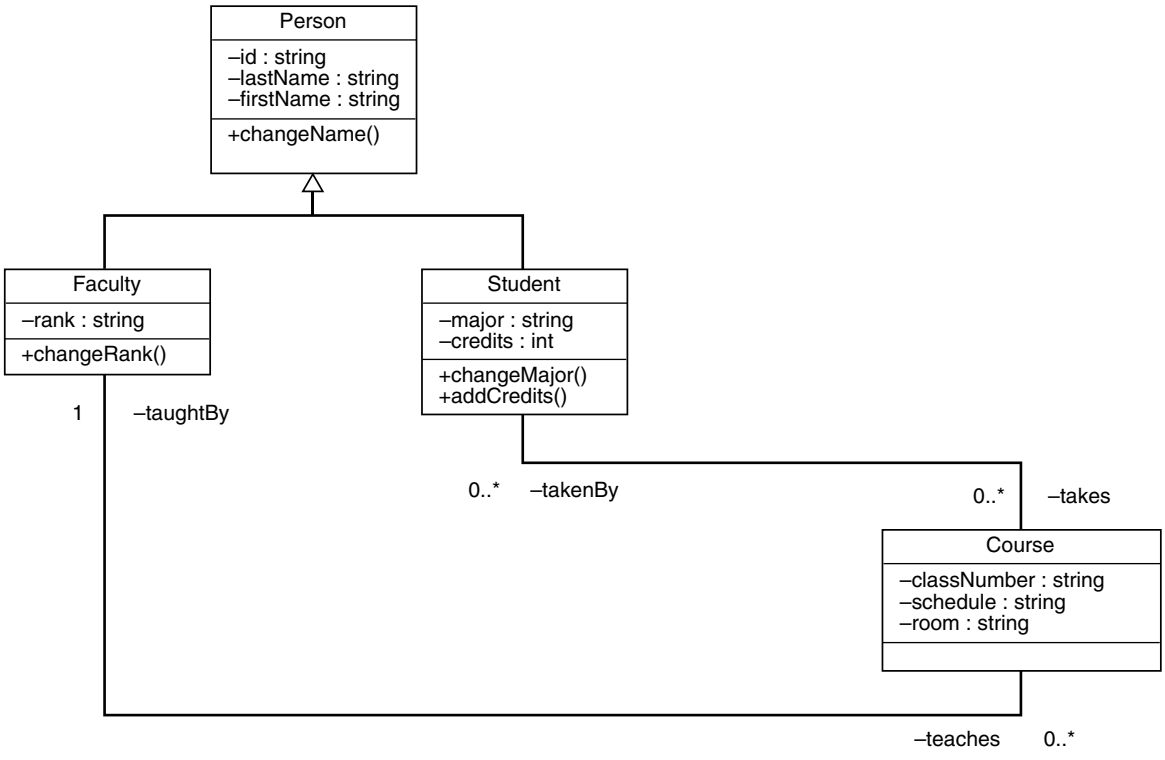


Figure 2.10
A Class Diagram

2.7.4 Object-Relational Model

The object-relational model extends the relational model by adding complex data types and methods as well as other advanced features. Instead of atomic, single-valued attributes as required in the relational model, this model allows attributes to be structured and have sets or arrays of values. It also permits methods and type inheritance. The SQL language was extended to create and manipulate the more complex data types and facilities that this model supports. Therefore the language used to manage an object-relational database is closer to the type of language used for relational databases than that used for strictly object-oriented databases.

2.7.5 Semistructured Data Model

Most data models require that entity sets (or classes or records, depending on the model) have the same structure. The structure is defined in the

schema, and remains unchanged unless the DBA changes the schema. By contrast, the semi-structured model allows a collection of nodes, each containing data, possibly with different schemas. The node itself contains information about the structure of its contents. Semi-structured databases are especially useful when existing databases having different schemas must be integrated. The individual databases can be viewed as documents, and XML (Extensible Markup Language) tags can be attached to each document to describe its contents. XML is a language similar to HTML (Hypertext Markup Language), but is used as a standard for data exchange rather than data presentation. XML tags are used to identify elements, sub-elements, and attributes that store data. The schema can be specified using a Document Type Definition (DTD) or by an XML schema that identifies the elements, their attributes, and their relationships to one another.

2.8 Chapter Summary

A corporation's operational data is a corporate resource, an asset that is of value to the organization and incurs cost. A database, which is shared by many users, protects operational data by providing data security, integrity constraints, reliability controls, and professional management by a DBA.

Data means facts, while information is processed data. There are four levels of discussion about data: **reality** (the real world) containing the **miniworld** (Universe of Discourse) that we are modeling, the abstract **conceptual model** of the miniworld, the **logical model** or **intention** of the database with its **metadata** (data about data), and the **database extension**, with **data instances**.

A staged approach to database design is a top-down approach that allows the designer to develop a conceptual model that mirrors the operations of the organization, allows changes, supports many user views, is independent of physical implementation, and does not depend on the model of a particular DBMS. This approach allows the database to evolve as needs change. In staged database design the designer must analyze the user environment, develop a conceptual data model, choose the DBMS, create the logical model by mapping the conceptual model to the data model of the DBMS, develop the internal and physical models, evaluate the physical

model, perform tuning if needed, and implement the physical model. Steps may be repeated until the design is satisfactory.

Design methodologies can be general techniques or commercial products such as **CASE** packages. A **data dictionary** can be **integrated** (the DBMS system catalog) or **freestanding** (independent of a DBMS). A data dictionary is a valuable tool for collecting and organizing information about data.

The **DBA** must be technically competent, a good manager, and have excellent interpersonal and communication skills. He or she has primary responsibility for planning, designing, developing, and managing the operating database.

Standard database architecture uses three levels of abstraction: **external**, **logical**, and **internal**. An **external view** is the model seen by a particular user. An **external schema** is an external model description written in the **data definition language**, which is part of the **data sublanguage** for the DBMS being used. The **user interface** creates the user's working environment and hides lower levels from the user. The **logical schema** is a complete DDL description of the logical model. It specifies the information content of the database, including all record types and fields. It is derived from the conceptual model. A good conceptual model results in a logical model that is easy to implement and supports the desired external views. The **logical record interface** is a boundary below which the logical level cannot see. The **internal schema** is a DDL description of the internal model. It specifies how data is represented, how records are sequenced, what indexes and pointers exist, and what hashing scheme is used. The **stored record interface** is the boundary below which the DBMS does not see. The operating system should be responsible for all physical details below this interface, including creating a **physical record interface** for itself to handle low-level physical details such as track placement. The **external/logical mapping** tells how the external views correspond to the logical items. The **logical/internal** mapping tells how logical items correspond to internal ones. **Data independence** makes each level immune to changes on lower levels. **Logical data independence** means that the logical level can be changed without changing external views. **Physical data independence** means internal and physical changes will not affect the logical model.

Some data models are the **entity-relationship**, **object-oriented**, **object-relational**, and **semistructured** models. There are also **record-based**

models, including the **relational** as well as the older **network** and **hierarchical** models. The entity-relationship model uses **E-R diagrams** to show entity sets, attributes of entities, relationship sets, and descriptive attributes of relationships. The relational model uses **tables** to represent data and relationships. The object-oriented model uses the concept of a **class**, having attributes and methods. An **object** is created as an instance of a class. Object-oriented databases contain **persistent** objects. Object-relational databases are extensions of relational model databases to allow complex objects and methods. Semistructured databases consist of nodes that are self-describing using XML.

Exercises

2.1 Name four resources of a typical business organization.

2.2 Distinguish between data and information.

2.3 Identify the four levels of abstraction in discussing data. For each, give an example of an item that appears on that level.

2.4 Distinguish between an entity set and an entity instance.

2.5 Distinguish between a record type and a record occurrence.

2.6 What level of data abstraction is represented in the data dictionary? Give examples of the types of entries that would be stored there.

2.7 Explain why a staged design approach is appropriate for database design.

2.8 Name five desirable characteristics of a conceptual model of an enterprise.

2.9 Name the eight major design stages in staged database design, along with the activities and possible outcomes of each.

2.10 Explain what is meant by saying staged database design is a "top-down" method.

2.11 Explain how a CASE package can be used in database design.

2.12 Name two advantages and two disadvantages of the following:

a. integrated data dictionaries

b. freestanding data dictionaries

2.13 Explain why users should not have access to the data dictionary.

2.14 Name eight uses for a data dictionary.

2.15 What types of skills must a database administrator possess? For what tasks are they needed?

2.16 List the major functions of the DBA.

2.17 Define each of the following terms:

a. operational data

b. corporate resource

c. metadata

d. entity

e. attribute

f. data item

g. data aggregate

h. data record

i. data file

j. data sublanguage

k. prototype

l. system tuning

m. CASE

n. integrated data dictionary

o. data dictionary synonym

p. data dictionary homonym

q. data standards

r. DBA

2.18 Describe the three-level architecture for databases.

2.19 Describe the two parts of data sublanguages.

2.20 Give five reasons why it is desirable to separate the physical representation from the external structure of a database.

2.21 Distinguish between the following: user interface, logical record interface, stored record interface, physical record interface.

2.22 Describe each of the following: external schema, logical schema, internal schema.

2.23 Explain the purpose of the external/logical mapping and of the logical/internal mapping.

2.24 Distinguish between logical and physical data independence, and give examples of possible changes they allow.

2.25 Explain why the logical model is called the heart of the database.

2.26 Describe abstraction and explain how it is used in database design.

2.27 Distinguish between the intension and extension of a database.

2.28 Explain how record-based data models differ from semantic models.

On the Companion Website:

- Lab Exercises
- Sample Project
- Student Projects