3.1. Data Modeling and Data Models

Wednesday, September 21, 2022, 2:17 PM

Number of replies: 3

Database design focuses on how the database structure will be used to store and manage end-user data. Data modeling, the first step in designing a database, refers to the process of creating a specific data model for a determined problem domain. (A problem domain is a clearly defined area within the real-world environment, with a well-defined scope and boundaries that will be systematically addressed.) A data model is a relatively simple representation, usually graphical, of more complex real-world data structures. In general terms, a model is an abstraction of a more complex real-world object or event. A model's main function is to help you understand the complexities of the real-world environment. Within the database environment, a data model represents data structures and their characteristics, relations, constraints, transformations, and other constructs with the purpose of supporting a specific problem domain.

Data modeling is an iterative, progressive process. You start with a simple understanding of the problem domain, and as your understanding increases, so does the level of detail of the data model. When done properly, the final data model effectively is a "blueprint" with all the instructions to build a database that will meet all end-user requirements. This blueprint is narrative and graphical in nature, meaning that it contains both text descriptions in plain, unambiguous language and clear, useful diagrams depicting the main data elements.

3.2. The Importance of Data Models

Wednesday, September 21, 2022, 2:17 PM

Number of replies: 3

Data models can facilitate interaction among the designer, the applications programmer, and the end-user. A well-developed data model can even foster an improved understanding of the organization for which the database design is developed. In short, data models are a communication tool. This important aspect of data modeling was summed up neatly by a client whose reaction was as follows: "I created this business, I worked with this business for years, and this is the first time I've really understood how all the pieces really fit together."

The importance of data modeling cannot be overstated. Data constitute the most basic information employed by a system. Applications are created to manage data and to help transform data into information, but data is viewed in different ways by different people. For example, contrast the view of a company manager with that of a company clerk. Although both work for the same company, the manager is more likely to have an enterprise-wide view of company data than the clerk.

Even different managers view data differently. For example, a company president is likely to take a universal view of the data because he or she must be able to tie the company's divisions to a common (database) vision. A purchasing manager in the same company is likely to have a more restricted view of the data, as is the company's inventory manager. In effect, each department manager works with a subset of the company's data. The inventory manager is more concerned about inventory levels, while the purchasing manager is more concerned about the cost of items and about relationships with the suppliers of those items.

Applications programmers have yet another view of data, being more concerned with data location, formatting, and specific reporting requirements. Basically, application programmers translate company policies and procedures from a variety of sources into appropriate interfaces, reports, and query screens.

The different users and producers of data and information often reflect the fable of the blind people and the elephant: the blind person who felt the elephant's trunk had quite a different view from the one who felt the elephant's leg or tail. A view of the whole elephant is needed. Similarly, a house is not a random collection of rooms; to build a house, a person should first have the overall view that is provided by blueprints. Likewise, a sound data environment requires an overall database blueprint based on an appropriate data model.

When a good database blueprint is available, it does not matter that an applications programmer's view of the data is different from that of the manager or the end-user. Conversely, when a good database blueprint is not available, problems are likely to ensue. For instance, an inventory management program and an order entry system may use conflicting product-numbering schemes, thereby costing the company thousands or even millions of dollars.

Keep in mind that a house blueprint is an abstraction; you cannot live in the blueprint. Similarly, the data model is an abstraction; you cannot draw the required data out of the data model. Just as you are not likely to build a good house without a blueprint, you are equally unlikely to create a good database without first creating an appropriate data model.

3.3. Data Model Basic Building Blocks

Wednesday, September 21, 2022, 2:17 PM

Number of replies: 2

The basic building blocks of all data models are entities, attributes, relationships, and constraints. An entity is a person, place, thing, or event about which data will be collected and stored. An entity represents a particular type of object in the real world, which means an entity is "distinguishable"—that

is, each entity occurrence is unique and distinct. For example, a CUSTOMER entity would have many distinguishable customer occurrences, such as John Smith, Pedro Dinamita, and Tom Strickland. Entities may be physical objects, such as customers or products, but entities may also be abstractions, such as flight routes or musical concerts.

An attribute is a characteristic of an entity. For example, a CUSTOMER entity would be described by attributes such as customer last name, customer first name, customer phone number, customer address, and customer credit limit. Attributes are the equivalent of fields in file systems.

A relationship describes an association among entities. For example, a relationship exists between customers and agents that can be described as follows: an agent can serve many customers, and each customer may be served by one agent. Data models use three types of relationships: one-to-many, many-to-many, and one-to-one. Database designers usually use the shorthand notations 1:M or 1..*, M:N or *..*, and 1:1 or 1..1, respectively. (Although the M:N notation is a standard label for the many-to-many relationship, the label M:M may also be used.) The following examples illustrate the distinctions among the three relationships.

One-to-many (1:M or 1..*) relationship. A painter creates many different paintings, but each is painted by only one painter. Thus, the painter (the “one”) is related to the paintings (the “many”). Therefore, database designers label the relationship “PAINTER paints PAINTING” as 1:M. Note that entity names are often capitalized as a convention, so they are easily identified. Similarly, a customer (the “one”) may generate many invoices, but each invoice (the “many”) is generated by only a single customer. The “CUSTOMER generates INVOICE” relationship would also be labeled 1:M.

Many-to-many (M:N or *..*) relationship. An employee may learn many job skills, and each job skill may be learned by many employees. Database designers label the relationship “EMPLOYEE learns SKILL” as M:N. Similarly, a student can take many classes and each class can be taken by many students, thus yielding the M:N label for the relationship expressed by “STUDENT takes CLASS.”

One-to-one (1:1 or 1..1) relationship. A retail company’s management structure may require that each of its stores be managed by a single employee. In turn, each store manager, who is an employee, manages only a single store. Therefore, the relationship “EMPLOYEE manages STORE” is labeled 1:1.

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

One CUSTOMER can generate many INVOICEs.

Each of the many INVOICEs is generated by only one CUSTOMER.

A constraint is a restriction placed on the data. Constraints are important because they help to ensure data integrity. Constraints are normally expressed in the form of rules:

An employee's salary must have values that are between 6,000 and 350,000.

A student's GPA must be between 0.00 and 4.00.

Each class must have one and only one teacher.

How do you properly identify entities, attributes, relationships, and constraints?

The first step is to clearly identify the business rules for the problem domain you are modeling.

3.4. Business Rules

Wednesday, September 21, 2022, 2:17 PM

Number of replies: 3

When database designers go about selecting or determining the entities, attributes, and relationships that will be used to build a data model, they might start by gaining a thorough understanding of what types of data exist in an organization, how the data is used, and in what time frames it is used. But such data and information do not, by themselves, yield the required understanding of the total business. From a database point of view, the collection of data becomes meaningful only when it reflects properly defined business rules. A business rule is a brief, precise, and unambiguous description of a policy, procedure, or principle within a specific organization. In a sense, business rules are misnamed: they apply to any organization, large or small—a business, a government unit, a religious group, or a research laboratory—that stores and uses data to generate information.

Business rules derived from a detailed description of an organization's operations help to create and enforce actions within that organization's environment. Business rules must be rendered in writing and updated to reflect any change in the organization's operational environment.

Properly written business rules are used to define entities, attributes, relationships, and constraints. Any time you see relationship statements such as "an agent can serve many customers, and each customer can be served by only one agent," business rules are at work.

To be effective, business rules must be easy to understand and widely disseminated to ensure that every person in the organization shares a common interpretation of the rules. Business rules describe, in simple language, the main and distinguishing characteristics of the data as viewed by the company. Examples of business rules are as follows:

A customer may generate many invoices.

An invoice is generated by only one customer.

A training session cannot be scheduled for fewer than 10 employees or for more than 30 employees.

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

Discovering Business Rules

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

The process of identifying and documenting business rules is essential to database design for several reasons:

It helps to standardize the company's view of data.

It can be a communication tool between users and designers.

It allows the designer to understand the nature, role, and scope of the data.

It allows the designer to understand business processes.

It allows the designer to develop appropriate relationship participation rules and constraints and to create an accurate data model.

Of course, not all business rules can be modeled. For example, a business rule that specifies "no pilot can fly more than 10 hours within any 24-hour period" cannot be modeled in the database model directly. However, such a business rule can be represented and enforced by application software.

Translating Business Rules into Data Model Components

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

Customer and invoice are objects of interest for the environment and should be represented by their respective entities.

There is a generate relationship between customer and invoice.

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

How many instances of B are related to one instance of A?

How many instances of A are related to one instance of B?

For example, you can assess the relationship between student and class by asking two questions:

In how many classes can one student enroll? Answer: many classes.

How many students can enroll in one class? Answer: many students.

Therefore, the relationship between student and class is many-to-many (M:N). You will have many opportunities to determine the relationships between entities as you proceed through this book, and soon the process will become second nature.

Naming Conventions

During the translation of business rules to data model components, you identify entities, attributes, relationships, and constraints. This identification process includes naming the object in a way that makes

it unique and distinguishable from other objects in the problem domain. Therefore, it is important to pay special attention to how you name the objects you are discovering.

Entity names should be descriptive of the objects in the business environment and use terminology that is familiar to the users. An attribute name should also be descriptive of the data represented by that attribute. It is also a good practice to prefix the name of an attribute with the name or abbreviation of the entity in which it occurs. For example, in the CUSTOMER entity, the customer's credit limit may be called CUS_CREDIT_LIMIT. The CUS indicates that the attribute is descriptive of the CUSTOMER entity, while CREDIT_LIMIT makes it easy to recognize the data that will be contained in the attribute.

3.5. The Evolution of Data Models

Wednesday, September 21, 2022, 2:17 PM

Number of replies: 2

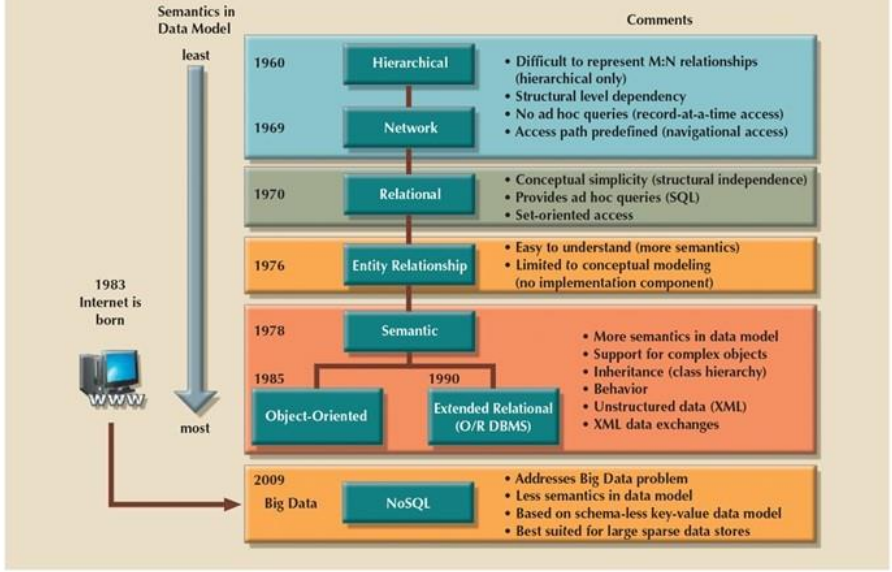
The quest for better data management has led to several models that attempt to resolve the previous model's critical shortcomings and to provide solutions to ever-evolving data management needs. These models represent schools of thought as to what a database is, what it should do, the types of structures that it should employ, and the technology that would be used to implement these structures. Perhaps confusingly, these models are called data models, as are the graphical data models. This section gives an overview of the major data models in roughly chronological order. You will discover that many of the "new" database concepts and structures bear a remarkable resemblance to some of the "old" data model concepts and structures. Table 2.1 traces the evolution of the major data models.

TABLE 2.1

EVOLUTION OF MAJOR DATA MODELS

GENERATION	TIME	DATA MODEL	EXAMPLES	COMMENTS
First	1960s–1970s	File system	VMS/VSAM	Used mainly on IBM mainframe systems Managed records, not relationships
Second	1970s	Hierarchical and network	IMS, ADABAS, IDS-II	Early database systems Navigational access
Third	Mid-1970s	Relational	DB2 Oracle MS SQL Server MySQL	Conceptual simplicity Entity relationship (ER) modeling and support for relational data modeling
Fourth	Mid-1980s	Object-oriented Object/relational (O/R)	Versant Objectivity/DB DB2 UDB Oracle 12c	Object/relational supports object data types Star Schema support for data warehousing Web databases become common
Fifth	Mid-1990s	XML Hybrid DBMS	dbXML Tamino DB2 UDB Oracle 12c MS SQL Server	Unstructured data support O/R model supports XML documents Hybrid DBMS adds object front end to relational databases Support large databases (terabyte size)
Emerging Models: NoSQL	Early 2000s to present	Key-value store Column store	SimpleDB (Amazon) BigTable (Google) Cassandra (Apache) MongoDB Riak	Distributed, highly scalable High performance, fault tolerant Very large storage (petabytes) Suited for sparse data Proprietary application programming interface (API)

FIGURE 2.5 THE EVOLUTION OF DATA MODELS



3.6. Degrees of Data Abstraction

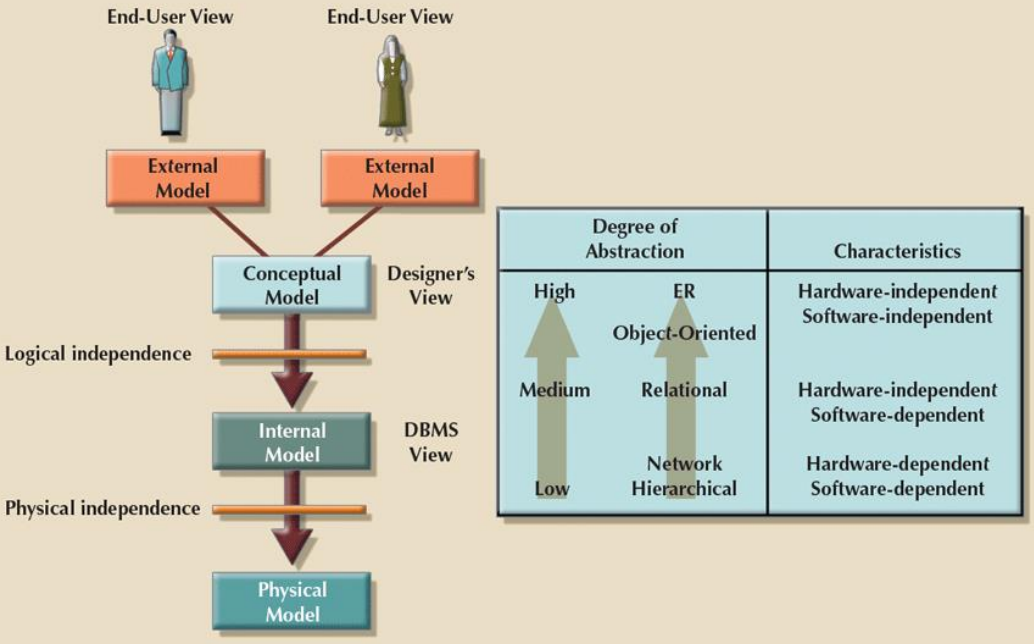
Wednesday, September 21, 2022, 2:17 PM

Number of replies: 3

If you ask 10 database designers what a data model is, you will end up with 10 different answers—depending on the degree of data abstraction. To illustrate the meaning of data abstraction, consider the example of automotive design. A car designer begins by drawing the concept of the car to be produced. Next, engineers design the details that help transfer the basic concept into a structure that can be produced. Finally, the engineering drawings are translated into production specifications to be used on the factory floor. As you can see, the process of producing the car begins at a high level of abstraction and proceeds to an ever-increasing level of detail. The factory floor process cannot proceed unless the engineering details are properly specified, and the engineering details cannot exist without the basic conceptual framework created by the designer. Designing a usable database follows the same basic process. That is, a database designer starts with an abstract view of the overall data environment and adds details as the design comes closer to implementation. Using levels of abstraction can also be very helpful in integrating multiple (and sometimes conflicting) views of data at different levels of an organization.

In the early 1970s, the American National Standards Institute (ANSI) Standards Planning and Requirements Committee (SPARC) defined a framework for data modeling based on degrees of data abstraction. The resulting ANSI/SPARC architecture defines three levels of data abstraction: external, conceptual, and internal. You can use this framework to better understand database models, as shown in Figure 2.6. In the figure, the ANSI/SPARC framework has been expanded with the addition of a physical model to explicitly address physical-level implementation details of the internal model.

FIGURE 2.6 DATA ABSTRACTION LEVELS



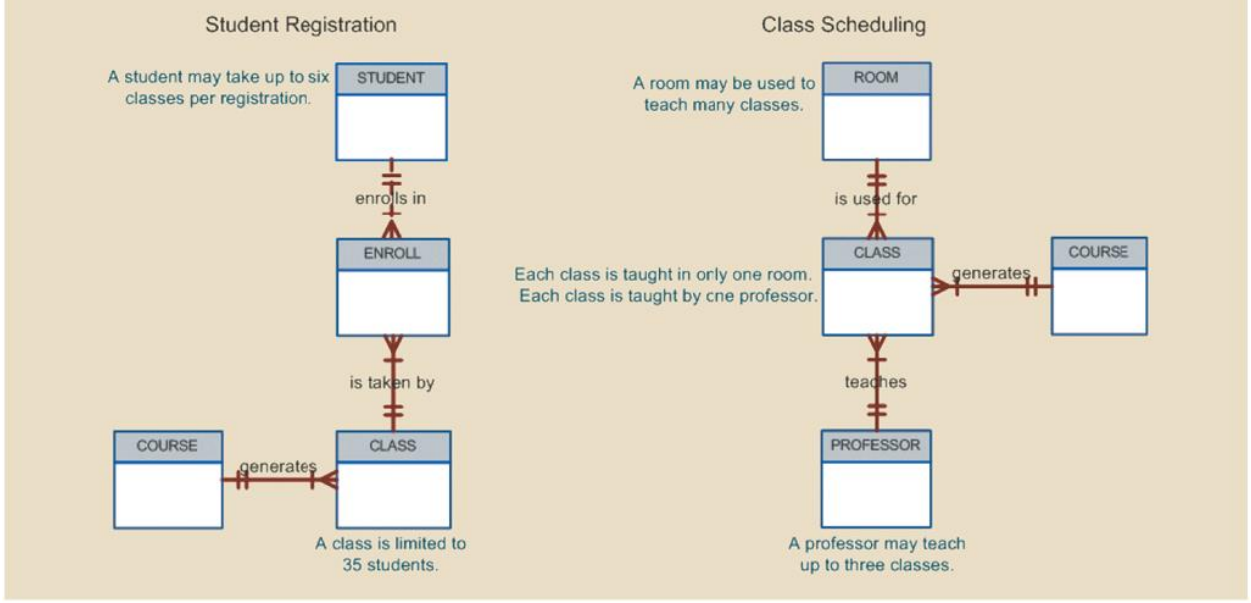
The External Model

The external model is the end users' view of the data environment. The term end users refers to people who use the application programs to manipulate the data and generate information. End users usually operate in an environment in which an application has a specific business unit focus. Companies are generally divided into several business units, such as sales, finance, and marketing. Each business unit is subject to specific constraints and requirements, and each one uses a subset of the overall data in the organization. Therefore, end users within those business units view their data subsets as separate from or external to other units within the organization.

Because data is being modeled, ER diagrams will be used to represent the external views. A specific representation of an external view is known as an external schema. To illustrate the external model's view, examine the data environment of Tiny College.

Figure 2.7 presents the external schemas for two Tiny College business units: student registration and class scheduling. Each external schema includes the appropriate entities, relationships, processes, and constraints imposed by the business unit. Also note that although the application views are isolated from each other, each view shares a common entity with the other view. For example, the registration and scheduling external schemas share the entities CLASS and COURSE.

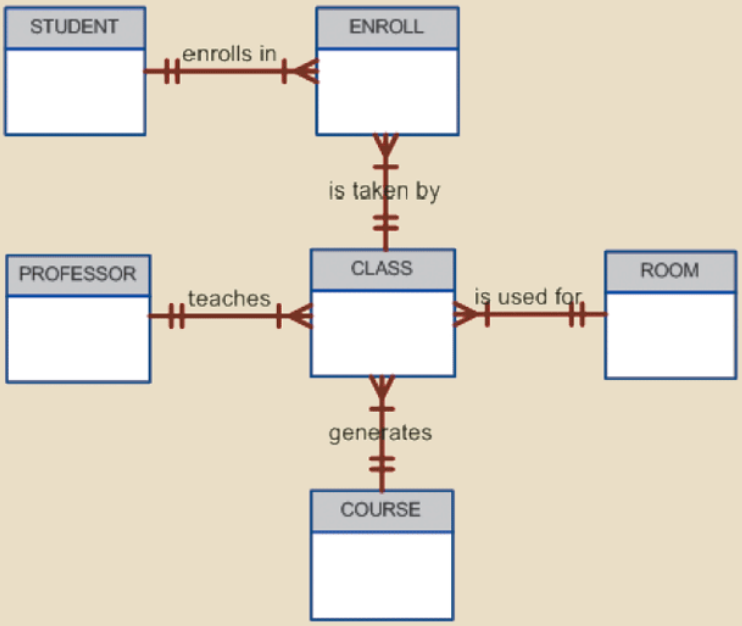
FIGURE 2.7 EXTERNAL MODELS FOR TINY COLLEGE



The Conceptual Model

The conceptual model represents a global view of the entire database by the entire organization. That is, the conceptual model integrates all external views (entities, relationships, constraints, and processes) into a single global view of the data in the enterprise, as shown in Figure 2.8. Also known as a conceptual schema, it is the basis for the identification and high-level description of the main data objects (avoiding any database model-specific details).

FIGURE 2.8 A CONCEPTUAL MODEL FOR TINY COLLEGE



The most widely used conceptual model is the ER model. Remember that the ER model is illustrated with the help of the ERD, which is effectively the basic database blueprint. The ERD is used to graphically represent the conceptual schema.

The conceptual model yields some important advantages. First, it provides a bird'seye (macro level) view of the data environment that is relatively easy to understand. For example, you can get a summary of Tiny College's data environment by examining the conceptual model in Figure 2.8.

Second, the conceptual model is independent of both software and hardware. Software independence means that the model does not depend on the DBMS software used to implement the model. Hardware independence means that the model does not depend on the hardware used in the implementation of the model. Therefore, changes in either the hardware or the DBMS software will have no effect on the database design at the conceptual level. Generally, the term logical design refers to the task of creating a conceptual data model that could be implemented in any DBMS.

The Internal Model

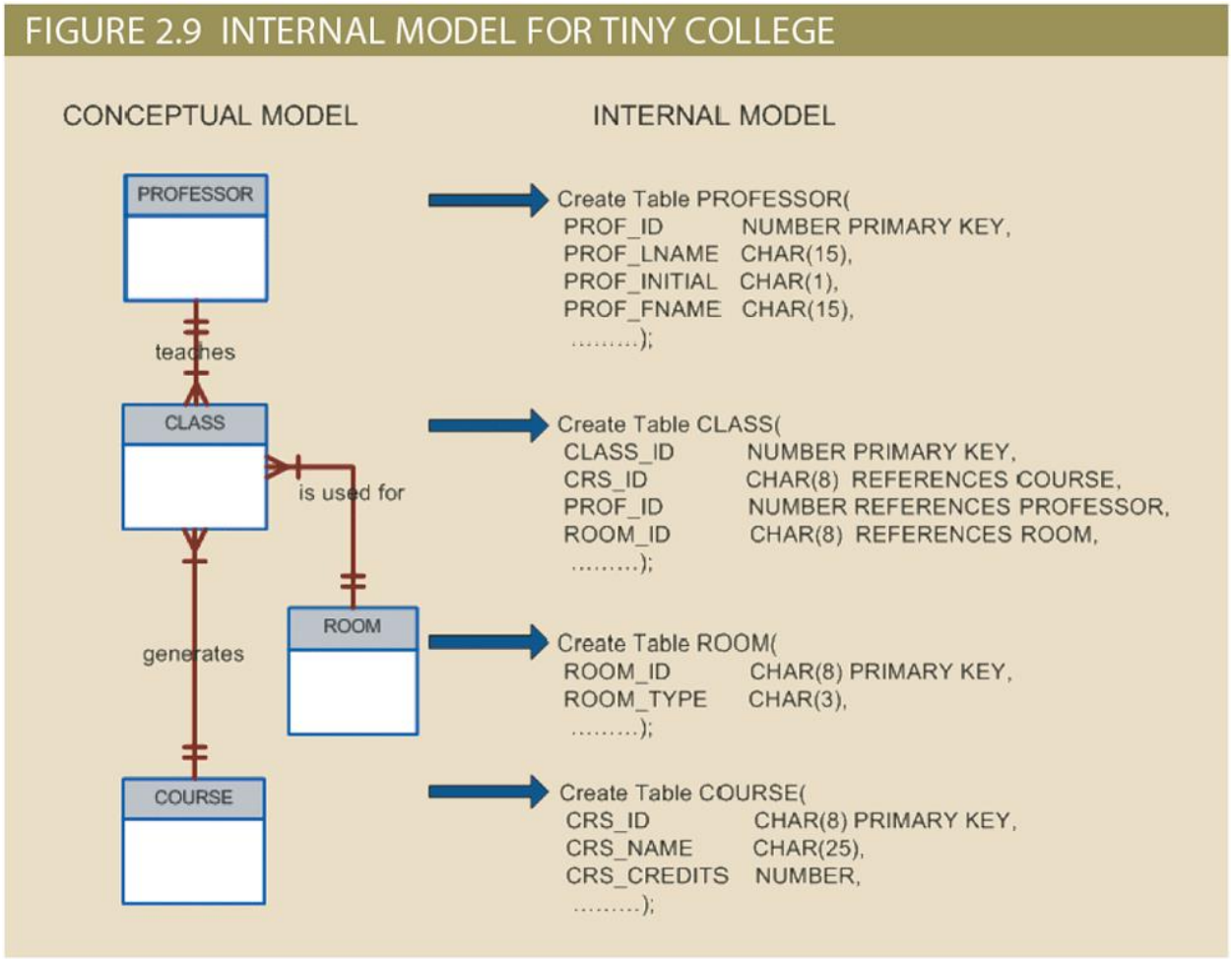
Once a specific DBMS has been selected, the internal model maps the conceptual model to the DBMS. The internal model is the representation of the database as "seen" by the DBMS. In other words, the

internal model requires the designer to match the conceptual model's characteristics and constraints to those of the selected implementation model.

An internal schema depicts a specific representation of an internal model, using the database constructs supported by the chosen database.

Because this book focuses on the relational model, a relational database was chosen to implement the internal model. Therefore, the internal schema should map the conceptual model to the relational model constructs. In particular, the entities in the conceptual model are mapped to tables in the relational model. Likewise, because a relational database has been selected, the internal schema is expressed using SQL, the standard language for relational databases. In the case of the conceptual model for Tiny College depicted in Figure 2.8, the internal model was implemented by creating the tables PROFESSOR, COURSE, CLASS, STUDENT, ENROLL, and ROOM. A simplified version of the internal model for Tiny College is shown in Figure 2.9.

FIGURE 2.9 INTERNAL MODEL FOR TINY COLLEGE



The development of a detailed internal model is especially important to database designers who work with hierarchical or network models because those models require precise specification of data storage

location and data access paths. In contrast, the relational model requires less detail in its internal model because most RDBMSs handle data access path definition transparently; that is, the designer need not be aware of the data access path details. Nevertheless, even relational database software usually requires specifications of data storage locations, especially in a mainframe environment.

For example, DB2 requires that you specify the data storage group, the location of the database within the storage group, and the location of the tables within the database.

Because the internal model depends on specific database software, it is said to be software dependent. Therefore, a change in the DBMS software requires that the internal model be changed to fit the characteristics and requirements of the implementation database model. When you can change the internal model without affecting the conceptual model, you have logical independence. However, the internal model is still hardware independent because it is unaffected by the type of computer on which the software is installed. Therefore, a change in storage devices or even a change in operating systems will not affect the internal model.

The Physical Model

The physical model operates at the lowest level of abstraction, describing the way data is saved on storage media such as magnetic, solid state, or optical media. The physical model requires the definition of both the physical storage devices and the (physical) access methods required to reach the data within those storage devices, making it both software and hardware dependent. The storage structures used are dependent on the software (the DBMS and the operating system) and on the type of storage devices the computer can handle. The precision required in the physical model's definition demands that database designers have a detailed knowledge of the hardware and software used to implement the database design.

Early data models forced the database designer to take the details of the physical model's data storage requirements into account. However, the now-dominant relational model is aimed largely at the logical level rather than at the physical level; therefore, it does not require the physical-level details common to its predecessors.

Although the relational model does not require the designer to be concerned about the data's physical storage characteristics, the implementation of a relational model may require physical-level fine-tuning for increased performance. Fine-tuning is especially important when very large databases are installed in a mainframe environment, yet even such performance fine-tuning at the physical level does not require knowledge of physical data storage characteristics.

As noted earlier, the physical model is dependent on the DBMS, methods of accessing files, and types of hardware storage devices supported by the operating system.

When you can change the physical model without affecting the internal model, you have physical independence. Therefore, a change in storage devices or methods and even a change in operating system will not affect the internal model.

The levels of data abstraction are summarized in Table 2.4.

TABLE 2.4

LEVELS OF DATA ABSTRACTION

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