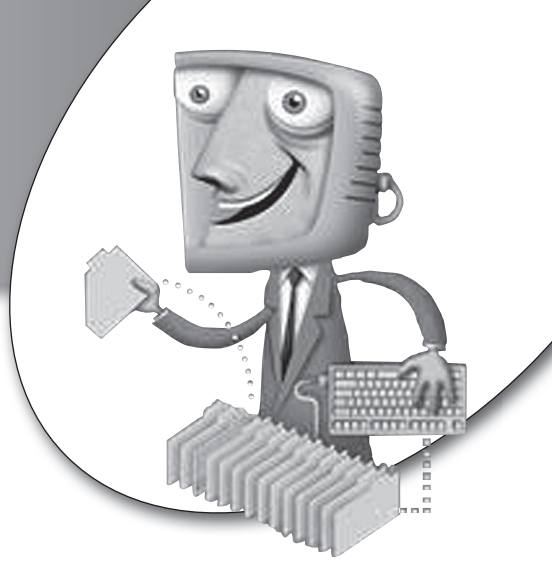


chapter 1

Database Fundamentals

This chapter introduces fundamental concepts and definitions regarding databases, including properties common to databases, prevalent database models, a brief history of databases, and the rationale for focusing on the relational model.

CHAPTER OBJECTIVES

In this chapter, the reader should:

- Understand the properties of a database and terms commonly used to describe databases.
- Identify the prevalent database models.
- Understand the history of databases.
- Explain why a focus on relational databases makes sense.

Properties of a Database

A *database* is a collection of interrelated data items that are managed as a single unit. This definition is deliberately broad because there is so much variety across the various software vendors that provide database systems. Microsoft Access places the entire database in a single data file, so an Access database can be defined as the file that contains the data items. Oracle Corporation defines their database as a collection of physical files that are managed by an instance of their database software product. A *file* is a collection of related records that are stored as a single unit by an operating system. An *instance* is a copy of the database software running in memory. Microsoft SQL Server and Sybase define a database as a collection of data items that have a common owner, and multiple databases are typically managed by a single instance of the database management software. This can be quite confusing if you work with multiple products because, for example, a database as defined by Microsoft SQL Server and Sybase is exactly what Oracle calls a *schema*.

A database is a collection of interrelated data items that are managed as a single unit.

Still Struggling

Given the unfortunately similar definitions of *files* and *databases,* how can we make a distinction? A number of Unix operating system vendors call their password file a "database," yet database experts will quickly point out that it is not. Clearly, we need a bit more rigor in our definitions. The answer lies in an understanding of certain characteristics or properties that databases possess that ordinary files do not, including management by a database management system (DBMS), layers of data abstraction, physical data independence, and logical data independence. These characteristics are discussed in subsections of this chapter.

A *database object* is a named data structure that is stored in a database. The specific types of database objects supported in a database vary from vendor to vendor and from one database model to another. *Database model* refers to the way

in which a database organizes its data to pattern the real world. The most common database models are presented in "Prevalent Database Models," later in this chapter.

The properties of databases are discussed in the following subsections.

The Database Management System (DBMS)

The *Database Management System (DBMS)* is software provided by the database vendor. Software products such as Microsoft Access, Oracle, Microsoft SQL Server, Sybase, DB2, Ingres, and MySQL are all DBMSs. (If it seems odd to you that the acronym used is "DBMS" instead of merely "DMS," keep in mind that the term "database" was originally written as two words and by convention has become a single compound word.)

The DBMS provides all the basic services required to organize and maintain the database, including the following:

- Moving data to and from the physical data files as needed
- Managing concurrent data access by multiple users including provisions to prevent simultaneous updates from conflicting with one another
- Managing transactions so that each transaction's database changes are an all-or-nothing unit of work. In other words, if the transaction succeeds, all database changes made by it are recorded in the database; if the transaction fails, none of the changes it made are recorded in the database
- Support for a *query language*, which is a system of commands that a database user employs to retrieve data from the database
- Provisions for backing up the database and recovering from failures
- Security mechanisms to prevent unauthorized data access and modification

Layers of Data Abstraction

What is unique about databases is that although they store the underlying data only once, they can present multiple users of the data with multiple distinct views of that data. These views are collectively called *user views*. A *user* in this context is any person or application that signs onto the database for the purpose of storing and/or retrieving data. An *application* is a set of computer programs designed to solve a particular business problem, such as an order-entry system, a payroll-processing system, or an accounting system.

TERMS: User Views

User views are abstractions provided by the DBMS that permit different users of the database to use customized presentations of the same data that are tailored to their exact needs. This property is one of the fundamental benefits that databases provide over simple file systems.

In contrast to a database, when an electronic spreadsheet application such as Microsoft Excel is used, all users must share a common view of the data that must match the way the data is physically stored in the underlying data file. If a user hides some columns in a spreadsheet, reorders the rows, and saves the spreadsheet, the next user who opens it will have the data presented in the manner in which the first user saved it. An alternative, of course, is for users to save their own copy in separate physical files, but then as one user applies updates, the other users' data becomes out of date. With database systems, we can present each user a view of the same data, but the views can be *tailored* to the needs of the individual users, even though the views all come from one commonly stored copy of the data. Because views store no actual data, they automatically reflect any data changes made to the underlying database objects. This is all possible through *layers of abstraction*, as shown in Figure 1-1.

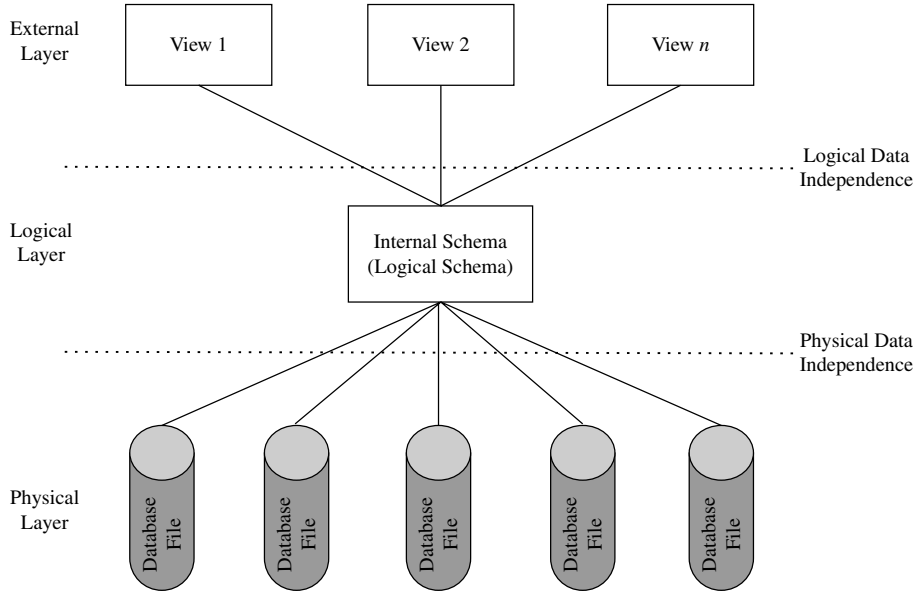


FIGURE 1-1 • Database layers of abstraction

The architecture shown in Figure 1-1 was first developed by ANSI/SPARC (American National Standards Institute Standards Planning and Requirements Committee) in the 1970s and quickly became a foundation for much of the database research and development efforts that followed. Most modern DBMSs follow this architecture, which is composed of three primary layers: the physical layer, the logical layer, and the external layer. The original architecture included a conceptual layer, which has been omitted here because none of the modern database vendors implemented it.

The Physical Layer

The *physical layer* contains the data files that hold all the data for the database. Nearly all modern DBMSs allow the database to be stored in multiple data files, which are usually spread out over multiple physical disk drives. With this arrangement, the disk drives can work in parallel for maximum performance. A notable exception is Microsoft Access, which stores the entire database in a single physical file. This arrangement limits the ability of the DBMS to scale to accommodate many concurrent users of the database, making it inappropriate as a solution for large enterprise systems, while simplifying database use on a single-user personal computer system.

The user of the database does not need to have any knowledge of how the data is actually stored within these files, or even which file contains the data item(s) of interest. In most organizations, a technician known as a *database administrator (DBA)* handles the details of installing and configuring the database software and data files and making the database available to the database users. The DBMS works with the computer's operating system to automatically manage the data files, including all file opening, closing, reading, and writing operations. The database user should not be required to refer to physical data files when using a database, which is in sharp contrast with spreadsheets and word processing, where the user must consciously save the document(s) and choose filenames and storage locations. Many of the personal computer-based DBMSs are exceptions to this tenet because the user is required to locate and open a physical file as part of the process of signing onto the DBMS. In contrast, with server-based DBMSs (such as Oracle, Sybase, Microsoft SQL Server, and so on), the physical files are managed automatically, and the database user never needs to refer to them when using the database.

The Logical Layer

The *logical layer* or *logical model* is the first of two layers of abstraction in the database. We say this because the physical layer has a concrete existence in the operating system files, whereas the logical layer exists only as abstract data structures assembled from the physical layer as needed. The DBMS transforms the data in the data files into a common structure. This layer is sometimes called the *schema*, a term used for the collection of all the data items stored in a particular database. (In some architectures, databases support multiple schemas. In this case, *schema* refers to all data items owned by a particular user account.) Depending on the particular DBMS, this can be a set of 2-D (two-dimensional) tables, a hierarchical structure similar to a company's organization chart, or some other structure. The "Prevalent Database Models" section later in this chapter describes the possible structures in more detail.

The External Layer

The *external layer* or *external model* is the second layer of abstraction in the database. This layer is composed of the user views discussed earlier, which are collectively called the *subschema*. This is the layer where users and application programs that access the database connect and issue queries against the database. Ideally, only the DBA deals with the physical layer, and only the DBA, developers, and other IT staff deal with the logical layers. The DBMS handles the transformation of selected items from one or more data structures in the logical layer to form each user view. The user views in this layer can be predefined and stored in the database for reuse, or they can be temporary items that are built by the DBMS to hold the results of a single ad hoc database query until no longer needed by the database user. By *ad hoc*, we mean a query that was not preconceived and one that is not likely to be reused. Views are discussed in more detail in Chapter 2.

Physical Data Independence

The ability to alter the physical file structure of a database without disrupting existing users and processes is known as *physical data independence*. As shown earlier in Figure 1-1, it is the separation of the physical layer from the logical layer that provides physical data independence in a DBMS. It is essential to understand that physical data independence is not a "have or have not" property, but rather one where a particular DBMS might have more or less data independence than another. The measure, sometimes called the *degree* of physical data independence, is how much change can be made in the file system without

impacting the logical layer. Prior to systems that offered data independence, even the slightest change to the way data was stored required the programming staff to make changes to every computer program that used the data, an expensive and time-consuming process.

TERMS: Physical Data Independence

Physical data independence is the ability to alter the physical file structure of a database without disrupting existing users and processes; such as moving database objects from one physical file to another.

All modern computer systems have some degree of physical data independence. For example, a spreadsheet on a personal computer will continue to work properly if copied from a hard disk to a USB thumb drive or if burned onto a CD. The fact that the performance (speed) of these devices varies is not the point, but rather that the devices have entirely different physical construction. Yet the operating system on the personal computer will automatically handle the differences and present the data in the file to the application (that is, the spreadsheet program, such as Microsoft Excel), and therefore to the user, in exactly the same way. However, on most personal systems, users must still remember where they placed the file so they can locate it when they need it again.

DBMSs expand greatly on the physical data independence provided by the computer system in that they allow database users to access database objects (for example, tables in a relational DBMS) without having to reference the physical data files in any way. The DBMS *catalog* keeps track of where the objects are physically stored. Here are some examples of physical changes that may be made in a data-independent manner:

- Moving a database data file from one device or directory to another
- Splitting or combining database data files
- Renaming database files
- Moving a database object from one data file to another
- Adding new database objects or data files

Note that we have made no mention of deleting things. It should be obvious that deleting a database object will cause anything that uses that object to fail. However, everything else should be unaffected.

Logical Data Independence

The ability to make changes to the logical layer without disrupting existing users and processes is called *logical data independence*. Figure 1-1, earlier in the chapter, shows that it is the transformation between the logical layer and the external layer that provides logical data independence. As with physical data independence, there are degrees of logical data independence. It is important to understand that most logical changes also involve a physical change. For example, you cannot add a new database object (such as a table in a relational DBMS) without physically storing the data somewhere; hence, there is a corresponding change in the physical layer. Moreover, deletion of objects in the logical layer will cause anything that uses those objects to fail, but should not affect anything else.

TERMS: Logical Data Independence

Logical data independence is the ability to make changes to the logical layer without disrupting existing users and processes, such as adding a new database object or adding a column to an existing database table.

Here are some examples of changes in the logical layer that can be safely made thanks to logical data independence:

- Adding a new database object
- Adding data items to an existing object
- Any change where a view can be placed in the external model that replaces (and processes the same as) the original object in the logical layer, such as combining or splitting existing objects

Prevalent Database Models

A *database model* is essentially the architecture that the DBMS uses to store objects within the database and to relate them to one another. (Be careful not to confuse the term "database model" with the term *data model*, which refers to the design of a particular database. You may find it helpful to think of database models as architectures used by the DBMS to store data, while data models are designs of specific databases such as order entry and payroll systems.)

The most prevalent database models are presented here in the order of their evolution. A brief history of relational databases appears in the next section to help put things in a chronological perspective.

Still Struggling

A bit more elaboration may help you understand the difference between database models and data models. A database model defines the architecture used by the DBMS much like a building code contains the regulations for constructing buildings. A data model, on the other hand, is a description of the design of an individual database, using both diagrams and text definitions, much like the blueprint for an individual building.

Flat Files

Flat files are "ordinary" operating system files in that records in the file contain no information to communicate the file structure or any relationship among the records to the application that uses the file. Any information about the structure or meaning of the data in the file must be included in each application that uses the file or must be known to each human who reads the file. In essence, flat files are not databases at all because they do not meet any of the criteria previously discussed. However, it is important to understand them for two reasons. First, flat files are often used to store database information. In this case, the operating system is still unaware of the contents and structure of the files, but the DBMS has metadata that allows it to translate between the flat files in the physical layer and the database structures in the logical layer. *Metadata*, which literally means "data about data," is the term used for the information that the database stores in its catalog to describe the data stored in the database and the relationships among the data. The metadata for a customer, for example, might include a list of all the data items collected about the customer, along with the length, minimum and maximum data values, and a brief description of each data item. Second, flat files existed before databases, and the earliest database systems *evolved* from the flat file systems that preceded them.

Figure 1-2 shows a sample flat file system, a subset of the data in the Microsoft Northwind sample database in this case. Northwind Traders is a supplier of international food items. Keep in mind that the column titles (Customer ID, Company Name, and so on) are included for illustration purposes only—only the data records would be stored in the actual files. Customer data is stored in a Customer file, with each record representing a Northwind customer. Each employee of Northwind has a record in the Employee file, and each product sold by Northwind has a record in the Product file. Order data (orders placed with Northwind by its customers) is stored in two other flat files. The Order file contains one record for each customer order with data about the orders, such as the customer ID of the customer who placed the order and the name of the employee who accepted the order from the customer. The Order Detail file contains one record for each line item on an order (an order can contain multiple line items, one for each product ordered), including data such as the unit price and quantity.

An *application program* is a unit of computer program logic that performs a particular function within an application system. Northwind has an application program that prints a listing of all the orders. This application must correlate

Customer File

Customer ID	Company Name	Contact First Name	Contact Last Name	Job Title
6	Company F	Francisco	Pérez-Olaeta	Purchasing Manager
26	Company Z	Run	Liu	Accounting Assistant

Employee File

Employee ID	First Name	Last Name	Title
2	Andrew	Cencini	Vice President, Sales
5	Steven	Thrope	Sales Manager
9	Anne	Hellung-Larsen	Sales Representative

Product File

Product ID	Product Code	Product Name	Category	Quantity Per Unit	List Price
5	NWTO-5	Northwind Traders Olive Oil	Oil	36 boxes	$21.35
7	NWTDFN-7	Northwind Traders Dried Pears	Dried Fruit & Nuts	12 – 1 lb pkgs.	$30.00
40	NWTCM-40	Northwind Traders Crab Meat	Canned Meat	24 – 4 oz tins	$18.40
41	NWTSO-41	Northwind Traders Clam Chowder	Soups	12 – 12 oz cans	$9.65
48	NWTCA-48	Northwind Traders Chocolate	Candy	10 pkgs.	$12.75
51	NWTDFN-51	Northwind Traders Dried Apples	Dried Fruit & Nuts	50 – 300 g pkgs.	$53.00

Order File

Order ID	Customer ID	Employee ID	Order Date	Shipped Date	Shipping Fee
51	26	9	4/5/2010	4/5/2010	$60.00
56	6	2	4/3/2010	4/3/2010	$0.00
79	6	2	6/23/2010	6/23/2010	$0.00

Order Detail File

Order ID	Product ID	Unit Price	Quantity
51	5	$21.35	15
51	41	$9.65	21
51	40	$18.40	2
56	48	$12.75	20
79	7	$30.00	14
79	51	$53.00	8

FIGURE 1-2 • Flat file order system

the data between the five files by reading an order and performing the following steps:

1. Use the customer ID to find the name of the customer in the Customer file.
2. Use the employee ID to find the name of the related employee in the Employee file.
3. Use the order ID to find the corresponding line items in the Order Detail file.
4. For each line item, use the product ID to find the corresponding product name in the Product file.

This is rather complicated given that we are just trying to print a simple listing of all the orders, yet this is the best possible data design for a flat file system.

One alternative design would be to combine all the information into a single data file. Although this would greatly simplify data retrieval, consider the ramifications of repeating all the customer data on every single order line item. You might not be able to add a new customer until they have an order ready to place. Also, if someone deletes the last order for a customer, you would lose all the information about the customer. But the worst situation is when customer information changes, because you have to find and update every record where the customer data is repeated. We will explore these issues much more deeply when we explore logical database design in Chapter 7.

Another alternative approach often used in flat file–based systems is to combine closely related files, such as the Order file and Order Detail file, into a single file, with the line items for each order following each order header record, and a Record Type data item added to help the application distinguish between the two types of records. Although this approach makes correlating the order data easier, it does so by adding the complexity of mixing two different kinds of records into the same file, so there is no net gain in either simplicity or faster application development.

Overall, the worst problem with the flat file approach is that the definition of the contents of each file and the logic required to correlate the data from multiple flat files have to be included in every application program that requires those files, thus adding to the expense and complexity of the application programs. It was this problem that provided computer scientists of the day with the incentive to find a better way to organize data.

The Hierarchical Model

The earliest databases followed the hierarchical model. The model evolved from the file systems that the databases replaced, with records arranged in a hierarchy much like an organization chart. Each file from the flat file system became a *record type*, or *node* in hierarchical terminology, but we will use the term *record* here for simplicity. Records were connected using *pointers* that contained the address of the related record. Pointers told the computer system where the related record was physically located, much as a street address directs us to a particular building in a city or a URL directs us to a particular web page or file on the Internet. Each pointer establishes a parent-child relationship, also called a *one-to-many relationship*, where one parent may have many children, but each child may have only one parent. This is similar to the situation in a traditional business organization, where each manager may have many employees as direct reports, but each employee may have only one manager. The obvious problem with the hierarchical model is that there is data that does not exactly fit this strict hierarchical structure, such as an order that must have the customer who placed the order as one parent and the employee who accepted the order as another. Data relationships are presented in more detail in Chapter 2. The most popular hierarchical database was Information Management System (IMS) from IBM.

Figure 1-3 shows the hierarchical structure of the hierarchical model for the Northwind database. You will recognize the Customer, Employee, Product, Order, and Order Detail record types introduced previously. Comparing the hierarchical structure with the flat file system shown in Figure 1-2, note that the Employee and Product records are shown in the hierarchical structure with dotted lines because they cannot be connected to the other records via pointers. These illustrate the most severe limitation of the hierarchical model that was the main reason for its early demise: no record may have more than one parent. Therefore, we *cannot* connect the Employee records with the Order records because the Order records already have the Customer record as their parent. Similarly, the Product records cannot be related to

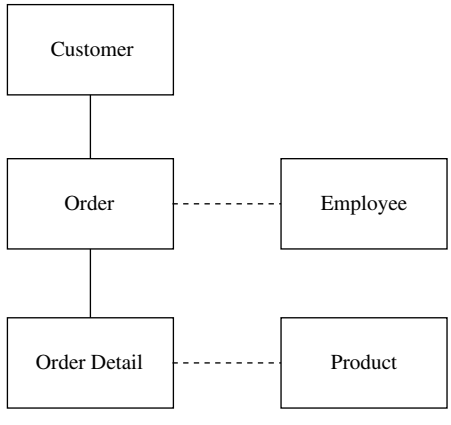


FIGURE 1-3 • Hierarchical model structure for Northwind

the Order Detail records because the Order Detail records already have the Order record as their parent. Database technicians had to work around this shortcoming either by relating the "extra" parent records in application programs, much as was done with flat file systems, or by repeating all the records under each parent, which of course was very wasteful of then-precious disk space. Neither of these was really an acceptable solution, so IBM modified IMS to allow for multiple parents per record. The resultant database model was dubbed the "Extended Hierarchical" model, which closely resembled the network database model in function, discussed in the next section.

Figure 1-4 shows the contents of selected records within the hierarchical model design for Northwind. For simplicity, only the identifiers of the records are shown, but a look back at Figure 1-2 should make the entire contents of each record clear to you. The record for Customer 6 has a pointer to its first order (ID 56), and that order has a pointer to the next order (ID 79). We know that Order 79 is the last order for the customer because it does not have a pointer to a subsequent order. Looking at the next layer in the hierarchy, Order 56 has a pointer to its only Order Detail record (for Product 48), while Order 79 has a pointer to its first Order Detail record (for Product 7), and that record has a pointer to the next detail record (for Product 51), and so forth. There is one additional important distinction between the flat file system and the hierarchical—the key (identifier) of the parent record is removed from the child records in the hierarchical model because the pointers handle the relationships among the records. Therefore, the Customer ID and Employee ID are removed from the Order record, and the Product ID is removed from the Order Detail record. Leaving them in is not a good idea because this could allow contradictory

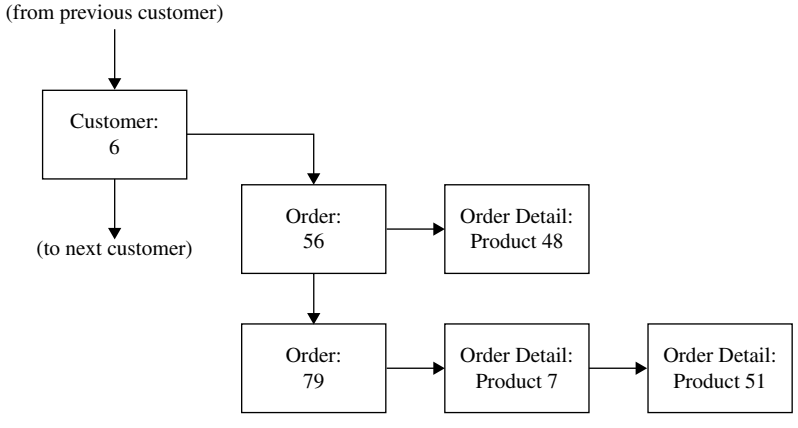


FIGURE 1-4 • Hierarchical model record contents for Northwind

information in the database, such as an order that is pointed to by one customer and yet contains the ID of a different customer.

The Network Model

The network database model evolved at around the same time as the hierarchical database model. A committee of industry representatives was formed to essentially build a better mousetrap. A cynic would say that a camel is a horse that was designed by a committee, and that may be accurate in this case. The most popular database based on the network model was the Integrated Database Management System (IDMS), originally developed by Cullinane (later renamed Cullinet). The product was enhanced with relational extensions, named IDMS/R, and eventually sold to Computer Associates.

As with the hierarchical model, record types (or simply "records") depict what would be separate files in a flat file system, and those records are related using one-to-many relationships, called *owner-member* relationships or *sets* in network model terminology. We'll stick with the terms *parent* and *child*, again for simplicity. As with the hierarchical model, physical address pointers are used to connect related records, and any identification of the parent record(s) is removed from each child record to avoid possible inconsistencies. In contrast with the hierarchical model, the relationships are named so the programmer can direct the database to use a particular relationship to navigate from one record to another in the database, thus allowing a record type to participate as the child in multiple relationships. The network model provided greater flexibility, but as is often the case with computer systems, at the expense of greater complexity.

The network model structure for Northwind, as shown in Figure 1-5, has all the same records as the equivalent hierarchical model structure that appeared in Figure 1-3. By convention, the arrowhead on the lines points from the parent record to the child record. Note that the Customer and Employee records now have solid lines in the structure diagram because they can be directly implemented.

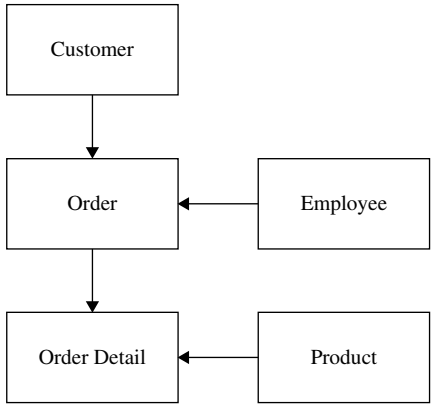


FIGURE 1-5 • Network model structure for Northwind

In the network model contents example shown in Figure 1-6, each parent-child

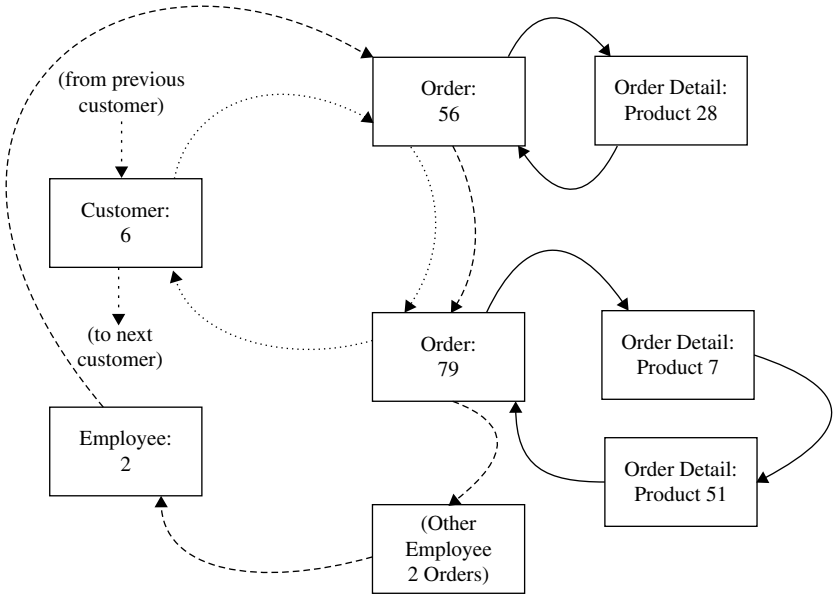


FIGURE 1-6 • Network model record contents for Northwind

relationship is depicted with a different type of line, illustrating that each has a different name. This difference is important because it points out the largest downside of the network model, which is complexity. Instead of a single path that may be used for processing the records, there are now many paths. For example, if we start with the record for Employee 2 and use it to find the first order (ID 56), we land in the chain of orders that belong to Customer 6. We happen to land on the first order belonging to Customer 6, but this is merely by chance—had there been orders for Customer 6 that were taken by other employees, we could have landed in the middle of the chain. To find all the other orders for this customer, there must be a way to work forward from where we are to the end of the chain and then wrap around to the beginning and forward from there until we return to the order from which we started. It is to satisfy this processing need that all pointer chains in network model databases are circular. As you might imagine, these circular pointer chains can easily result in an infinite loop (that is, a process that never ends) should database users not keep careful track of where they are in the database and how they got there. The structure of the Web loosely parallels a network database in that each web page has links to other related web pages, and circular references are not uncommon.

The process of navigating through a network database was called "walking the set" because it involved choosing paths through the database structure much like choosing walking paths through a forest when there can be multiple ways to get to the same destination. Without an up-to-date map, it is easy to get lost, or worse yet, to find a dead end where you cannot get to the desired destination record. The complexity of this model and the expense of the small army of technicians required to maintain it were key factors in its eventual demise.

The Relational Model

In addition to complexity, the network and hierarchical database models share another common problem—they are inflexible. You must follow the preconceived paths through the data in order to process the data efficiently. Ad hoc queries, such as finding all the orders shipped in a particular month, could require scanning the entire database to find them all. Computer scientists were still looking for a better way. Rarely in the history of computers has a development been truly revolutionary, but the research work of Dr. E.F. Codd that led to the relational model was clearly just that.

The relational model is based on the notion that any preconceived path through a data structure is too restrictive a solution, especially in light of ever-increasing demands to support ad hoc requests for information. Database users simply cannot think of every possible use of the data before the database is created; therefore, imposing predefined paths through the data merely creates a "data jail." The relational model therefore provides the ability to relate records *as needed* rather than predefining them when the records are first stored in the database. Moreover, the relational model is constructed such that queries can work with sets of data (for example, all the customers who have an outstanding balance) rather than one record at a time, as with the network and hierarchical models.

TERMS: **Relational Model**

The relational model is a database model that presents data in 2-D tables using common data to link tables. For example, a Customer ID stored in an order table can be used to link orders to the Customer table that contains information about the customers that placed the orders.

The relational model presents data in familiar 2-D tables, much like a spreadsheet does. Unlike with a spreadsheet, the data is not necessarily stored in tabular form, and the model also permits combining (*joining* in relational terminology) tables to form views, which are also presented as 2-D tables. In short, it follows the ANSI/SPARC model and therefore provides healthy doses of physical and logical data independence. Instead of linking related records together with physical address pointers, as is done in the hierarchical and network models, a common data item is stored in each table, just as was done in flat file systems.

Figure 1-7 shows the relational model design for Northwind. A look back at Figure 1-2 will confirm that each file in the flat file system has been mapped to a table in the relational model. As you will learn in Chapter 6, this one-to-one correspondence between flat files and relational tables will not always hold true, but it is quite common. In Figure 1-7, lines are drawn between the tables to show the one-to-many relationships, with the single-line end denoting the "one" side and the line end that splits into three parts (called a "crow's foot") denoting the "many" side. For example, merely by inspecting the lines that connect these tables, you can see that "one" customer is related to "many" orders and that "one" order is related to "many" order details. The diagramming technique shown here, called the *entity-relationship diagram* (ERD), will be covered in more detail in Chapter 7.

In Figure 1-8, three of the five tables have been represented with sample data in selected columns. In particular, note that the Customer ID column is stored in both the Customer table and the Order table. When the customer ID of a row in the Order table matches the customer ID of a row in the Customer table, you know that the order belongs to that particular customer. Similarly, the Employee ID

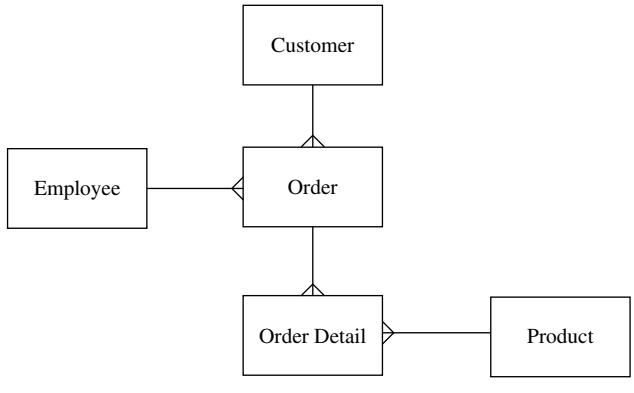
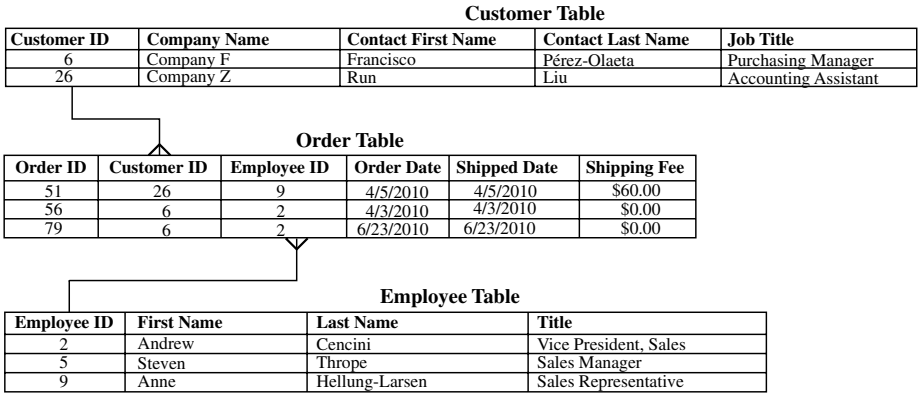


FIGURE 1-7 • Relational model structure for Northwind



Customer Table

Customer ID	Company Name	Contact First Name	Contact Last Name	Job Title
6	Company F	Francisco	Pérez-Olaeta	Purchasing Manager
26	Company Z	Run	Liu	Accounting Assistant

Order Table

Order ID	Customer ID	Employee ID	Order Date	Shipped Date	Shipping Fee
51	26	9	4/5/2010	4/5/2010	$60.00
56	6	2	4/3/2010	4/3/2010	$0.00
79	6	2	6/23/2010	6/23/2010	$0.00

Employee Table

Employee ID	First Name	Last Name	Title
2	Andrew	Cencini	Vice President, Sales
5	Steven	Thrope	Sales Manager
9	Anne	Hellung-Larsen	Sales Representative

FIGURE 1-8 • Relational table contents for Northwind

column is stored in both the Employee and Order tables to indicate the employee who accepted each order.

The elegant simplicity of the relational model and the ease with which people can learn and understand it has been the main factor in its universal acceptance. The relational model is the main focus of this book because it is ubiquitous in today's information technology systems and will likely remain so for many years to come.

The Object-Oriented Model

The object-oriented (OO) model actually had its beginnings in the 1970s, but it did not see significant commercial use until the 1990s. This sudden emergence came from the inability of then-existing RDBMSs (Relational Database Management Systems) to deal with complex data types such as images, complex drawings, and audio-video files. The sudden explosion of the Internet and the Web created a sharp demand for mainstream delivery of complex data.

An *object* is a logical grouping of related data and program logic that represents a real-world thing, such as a customer, employee, order, or product. Individual data items, such as customer ID and customer name, are called *variables* in the OO model and are stored within each object. In OO terminology, a *method* is a piece of application program logic that operates on a particular object and provides a finite function, such as checking a customer's credit limit or updating a customer's address. Among the many differences between the OO model and the models already presented, the most significant is that variables may *only* be accessed through methods. This property is called *encapsulation*.

The strict definition of *object* used here applies only to the OO model. The general term *database object*, as used earlier in this chapter, refers to any named item that might be stored in a non-OO database (for example, a table, index, or view). As OO concepts have found their way into relational databases, so has the terminology, although often with less precise definitions.

Figure 1-9 shows the Customer object as an example of OO implementation. The circle of methods around the central core of variables is to remind us of encapsulation. In fact, you can think of an object much like an atom with an electron field of methods and a nucleus of variables. Each customer for Northwind would have its own copy of the object structure, called an *object instance*, much as each customer has a copy of the customer record structure in the flat file system.

At a glance, the OO model looks horribly inefficient because it seems that each instance requires that the methods and the definition of the variables be redundantly stored. However, this is not at all the case. Objects are organized into a *class hierarchy* so that the common methods and variable definitions need only be defined once and then *inherited* by other members of the same class.

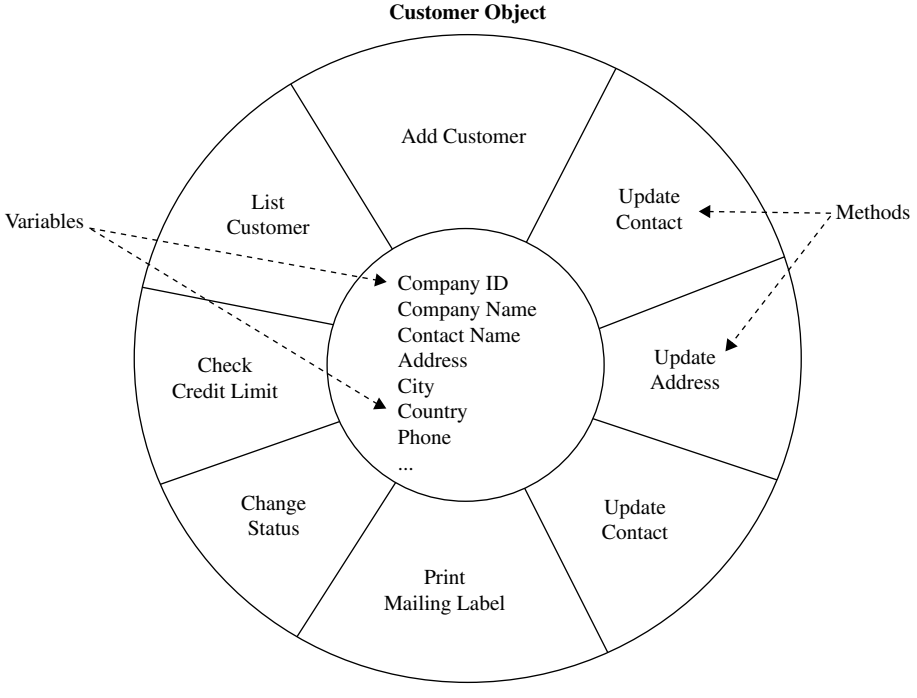


FIGURE 1-9 • The anatomy of an object

OO concepts have such benefit that they have found their way into nearly every aspect of modern computer systems. For example, the Microsoft Windows Registry has a class hierarchy.

The Object-Relational Model

Although the OO model provided some significant benefits in encapsulating data to minimize the effects of system modifications, the lack of ad hoc query capability has relegated it to a niche market where complex data is required, but ad hoc querying is not. However, some of the vendors of relational databases noted the significant benefits of the OO model and added object-like capability to their relational DBMS products with the hopes of capitalizing on the best of both models. The original name given to this type of database was "universal database," and although the marketing folks loved the term, it never caught on in technical circles, so the preferred name for the model became object-relational (OR). Through evolution, the Oracle, DB2, and Informix databases can all be said to be OR DBMSs to varying degrees.

To fully understand the OR model, a more detailed knowledge of the relational and OO models is required.

A Brief History of Databases

Space exploration projects led to many significant developments in the science and technology industries, including information technology. As part of the NASA Apollo moon project, North American Aviation (NAA) built a hierarchical file system named Generalized Update Access Method (GUAM) in 1964. IBM joined NAA to develop GUAM into the first commercially available hierarchical model database, called Information Management System (IMS), released in 1966.

Also in the mid-1960s, General Electric internally developed the first database based on the network model, under the direction of prominent computer scientist Charles W. Bachman, and named it Integrated Data Store (IDS). In 1967, the Conference on Data Systems Languages (CODASYL), an industry group, formed the Database Task Group (DBTG) and began work on a set of standards for the network model. In response to criticism of the "single parent" restriction in the hierarchical model, IBM introduced a version of IMS that circumvented the problem by allowing records to have one "physical" parent and multiple "logical" parents.

In June 1970, E.F. (Ted) Codd, an IBM researcher (later an IBM fellow), published a research paper titled "A Relational Model of Data for Large Shared Data Banks" in *Communications of the ACM*, the Journal of the Association for Computing Machinery, Inc. The publication can be easily found on the Internet. In 1971, the CODASYL DBTG published their standards, which were over three years in the making. This began five years of heated debate over which model was the best.

The CODASYL DBTG advocates argued the following:

- The relational model was too mathematical.
- An efficient implementation of the relational model could not be built.
- Application systems need to process data one record at a time.

The relational model advocates argued the following:

- Nothing as complicated as the DBTG proposal could possibly be the correct way to manage data.
- Set-oriented queries were too difficult in the DBTG language.
- The network model had no formal underpinnings in mathematical theory.

The debate came to a head at the 1975 ACM SIGMOD (Special Interest Group on Management of Data) conference. Ted Codd and two others debated against Charles Bachman and two others over the merits of the two models. At the end, the audience was more confused than beforehand. In retrospect, this happened because every argument proffered by the two sides was completely correct! However, interest in the network model waned markedly in the late 1970s. It was the evolution of database and computer technology that followed that proved the relational model was the better choice, including these significant developments:

- Query languages such as SQL emerged that were not so mathematical.
- Experimental implementations of the relational model proved that reasonable efficiency could be achieved, although never as efficient as an equivalent network model database. Also, computer systems continued to drop in price, and flexibility became more important than efficiency.
- Provisions were added to the SQL language to permit processing of a set of data using a record-at-a-time approach.
- Advanced tools made the relational model even easier to use.
- Codd's research led to the development of a new discipline in mathematics known as *relational calculus*.

In the mid-1970s, database research and development was at full steam. A team of 15 IBM researchers in San Jose, California, under the direction of Frank King, worked from 1974 to 1978 to develop a prototype relational database called System R. System R was built commercially and became the basis for HP ALLBASE and IDMS/SQL. Larry Ellison and a company that later became known as Oracle independently implemented the external specifications of System R. It is now common knowledge that Oracle's first customer was the CIA. With some rewriting, IBM developed System R into SQL/DS and then into DB2, which remains their flagship database to this day.

A pickup team of University of California, Berkeley, students under the direction of Michael Stonebraker and Eugene Wong worked from 1973 to 1977 to develop the Ingres DBMS. Ingres also became a commercial product and was quite successful. It is still available today as an open source solution.

In 1976, Dr. Peter Chen presented the entity-relationship (ER) model. His work bolstered the modeling weaknesses in the relational model and became the foundation of many modeling techniques that followed. If Ted Codd is considered the "father" of the relational model, then we must consider Peter Chen the "father" of the ER diagram. We explore ER diagrams in Chapter 7.

Sybase, which had a successful RDBMS deployed on Unix servers, entered into a joint agreement with Microsoft to develop the next generation of Sybase (to be called System 10) with a version available on Windows servers. For reasons not publicly known, the relationship soured before the products were completed, but each party walked away with all the work developed up to that point. Microsoft finished the Windows version and marketed the product as Microsoft SQL Server, whereas Sybase rushed to market with Sybase System 10. The products were so similar that instructors for Microsoft were known to use the more mature Sybase manuals in class rather than first-generation Microsoft documentation. The product lines have diverged considerably over the years, but Microsoft SQL Server's Sybase roots are still evident in the product.

Relational technology took the market by storm in the 1980s. Object-oriented databases, which first appeared in the 1970s, were also commercially successful during the 1980s. In the 1990s, object-relational systems emerged, with Informix being the first to market, followed relatively quickly by Oracle and IBM.

Not only did the relational technology of the day move around, but the people did also. Michael Stonebraker left UC Berkeley to found Illustra, an object-relational database vendor, and became chief science officer of Informix when it merged with Illustra. He is currently an adjunct professor at MIT, where he is involved in the development of a number of advanced database systems projects. Bob Epstein, who worked on the Ingres project with Stonebraker, moved to

the commercial company along with the Ingres product. From there he went to Britton-Lee (subsequently absorbed by NCR) to work on early *database machines* (computer systems with hardware and software specialized to run only databases) and then to start up Sybase, where he was the chief science officer for a number of years. Database machines, incidentally, died on the vine because they were so expensive compared with the combination of an RDBMS running on a general-purpose computer system. However, several vendors, including Oracle, Teradata, and Netezza, currently market database machines that use specialized software for running databases, but with industry-standard hardware. The San Francisco Bay Area was an exciting place for database technologists in that era, because all the great relational products started there, more or less in parallel, with the explosive growth of "Silicon Valley." Others have moved on, but Oracle and Sybase are still largely based in the Bay Area.

Why Focus on Relational?

The remainder of this book will focus on the relational model, with some coverage of the object-oriented and object-relational models. Aside from the relational model being the most prevalent of all the database models in modern business systems, there are other important reasons for this focus, especially for those learning about databases for the first time:

- Definition, maintenance, and manipulation of data storage structures is easy.
- Data is retrieved through simple ad hoc queries.
- Data is well protected.
- Well-established ANSI (American National Standards Institute) and ISO (International Organization for Standardization) standards exist.
- There are many vendors from which to choose.
- Conversion between vendor implementations is relatively easy.
- RDBMSs are mature and stable products.

Summary

In this chapter, you learned the properties of databases, terms used to describe databases, the prevalent database models, a brief history of databases, and the reasoning behind a focus on relational databases. In Chapter 2, we will explore the components of relational databases.

QUIZ

Choose the correct responses in each of the multiple-choice questions. Note that there may be more than one correct response to each question.

1. Some of the properties of a database are

A. It provides less logical data independence than the file systems it replaced.
B. It provides both physical and logical data independence.
C. Data items are stored exactly the way they are presented to the database user.
D. It provides layers of database abstraction.
E. Databases are always managed by a Database Management System.

2. Flat file systems:

A. Require the user or application program to relate one file to another
B. Require the user or application to know the contents of each file
C. Are not really databases by themselves, even though some vendors call them that
D. Provide no logical data independence when used directly by application programs
E. Can be used to store the database objects for a database

3. The hierarchical database model:

A. Stores data and methods together in the database
B. Was first developed by Dr. Peter Chen
C. In its pure form, permits only one parent for any given record
D. Connects data in a hierarchical structure using physical address pointers
E. Allows the processing of sets of database records

4. The network database model:

A. Allows the processing of sets of database records
B. Allows multiple parents for any given database record
C. Was first proposed by Dr. E.F. Codd
D. Is known for its simplicity of use
E. Connects database records using physical address pointers

5. The object-oriented model:

A. Was first invented in the 1980s
B. Stores data as variables along with application logic modules called "methods"
C. Restricts access to variables through encapsulation
D. Provides for freeform ad hoc querying of variables
E. Provides better support for complex data types than the relational model

6. The physical layer of the ANSI/SPARC model:

A. Provides physical data independence
B. Contains the physical files that comprise the database
C. Contains files that are read and written by the DBMS independently of the computer's operating system
D. Is normally invisible to the database user
E. Supplies data to the logical layer

7. The logical layer of the ANSI/SPARC model:

A. Contains database objects that are assembled by the DBMS from data in the physical layer
B. Contains the database schema
C. Lies between the physical and external layers
D. Provides logical data independence
E. Is referenced by the external layer

8. According to advocates of the relational model, the problems with the CODASYL model are

A. Set-oriented queries are too difficult.
B. An efficient implementation cannot be built.
C. It is too mathematical.
D. It is too complicated.
E. It lacks generally accepted standards.

9. According to the advocates of the network model, the problems with the relational model are

A. An efficient implementation cannot be built.
B. Record-at-a-time processing is poorly supported.
C. It has no formal mathematical underpinnings.
D. It is too complicated.
E. It lacks generally accepted standards.

10. Important historic events in database development are

A. Early relational databases were built by both IBM and UC Berkeley.
B. Nearly all the commercial relational databases are descendents of either System R or Ingres.
C. GUAM was the first commercially available database.
D. Dr. E.F. Codd published his famous research paper in 1970.
E. General Electric's IDS was the first known network database.