Chapter 1
Database Fundamentals

Chapter One

Database Fundamentals

1.1 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 so much variety exists across the various software vendors that provide database systems. For example, 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 its database as a collection of physical files that are managed by an instance of its database software product. An *instance* is a copy of the database software running in memory.

Microsoft SQL Server and Sybase Adaptive Server Enterprise (ASE) 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 all be quite confusing if you work with multiple products, because, for example, a database as defined by Microsoft SQL Server or Sybase ASE is exactly what Oracle Corporation calls a *schema*.

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.

1.2 The Database Management System

The *database management system* (DBMS) is software provided by the database vendor. Software products such as Microsoft Access, Oracle, Microsoft SQL Server, Sybase ASE, DB2, Ingres, and MySQL are all DBMSs.

If it seems odd to you that the DBMS acronym is used instead of merely DMS, remember that the term database was originally written as two words, and by convention has since become a single compound word.

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

- Moves data to and from the physical data files as needed.
- Manages concurrent data access by multiple users, including provisions to prevent simultaneous updates from conflicting with one another.
- Manages 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.

- Ω Supports a *query language,* which is a system of commands that a database user employs to retrieve data from the database.
- Ω Provides provisions for backing up the database and recovering from failures.
- Ω Provides security mechanisms to prevent unauthorized data access and modification.

A data bank and a database are the same thing. Data bank is merely an older term that was used by the scientists who developed early database systems. In fact, the term data bank is still used in a few human languages.

1.3 Layers of Data Abstraction

Databases are unique in their ability to present multiple users with their own distinct views of the data while storing the underlying data only once. These are collectively called user views.

A *user* in this context is any person or application that signs on to the database for the purpose of storing and/or retrieving data.

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.

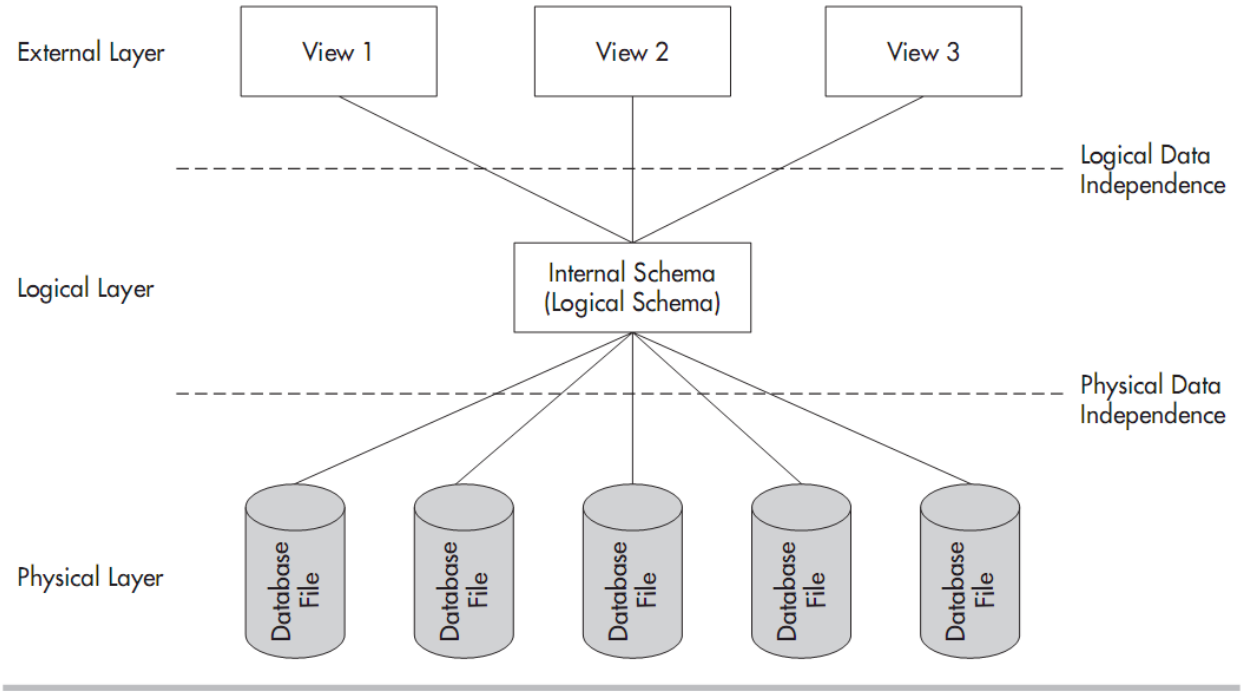

Figure 1-1: Database layers of abstraction

The Physical Layer

The *physical layer* contains the data files that hold all the data for the database.

The Logical Layer

The *logical layer* or *logical model* comprises the first of two layers of abstraction in the database: 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 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, which are collectively called the *subschema*.

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*. Here are some examples of physical changes that can be made in a data-independent manner:

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

Logical Data Independence

The ability to make changes to the logical layer without disrupting existing users and processes is called *logical data independence*. 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
- Making any change in which 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

1.4 Prevalent Database Models

A *database model* is essentially the architecture that the DBMS uses to store objects within the database and relate them to one another.

Flat Files

Flat files are "ordinary" operating system files, in that records in a file contain no information to communicate the file structure or any relationship among the records to the application that uses the file.

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 all the data items collected about the customer (such as name, address, and account status), along with the length, minimum and maximum data values, and a brief description of each data item, figure 1-2 shows a sample flat file system.

An *application program* is a unit of computer program logic that performs a particular function within an application system.

Customer File

Customer ID	Company Name	Contact First Name	Contact Last Name	Job Title	City	State
6	Company F	Francisco	Pérez-Olaeta	Purchasing Manager	Milwaukee	WI
26	Company Z	Run	Liu	Accounting Assistant	Miami	FL

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

Figure 1-2: Flat file order system

The Hierarchical Model

The earliest databases followed the hierarchical model, which 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 the term *record* is used 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 you to a particular building in a city, a URL directs you to a particular web page on the Internet, or GPS coordinates point to a particular location on the planet.

Each pointer establishes a parent-child relationship, also called a *one-to-many relationship*, in which one parent can have many children, but each child can have only one parent, figure 1-3 shows the hierarchical structure of the hierarchical model.

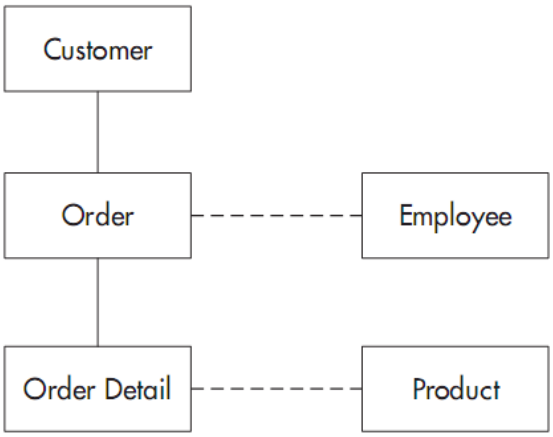

Figure 1-3: Hierarchical model structure

The Network Model

The network database model evolved at around the same time as the hierarchical database model. 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 DBMS 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 Relational Model

In addition to complexity, the network and hierarchical database models share another common problem—they are inflexible. Computer scientists were still looking for a better way.

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 allows users to relate records *as needed* rather than as predefined when the records are first stored in the database. Moreover, the relational model is constructed such that queries 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.

The relational model presents data in familiar two-dimensional tables, much like a spreadsheet does. Unlike 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 two-dimensional tables. 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 1-4 shows the relational model.

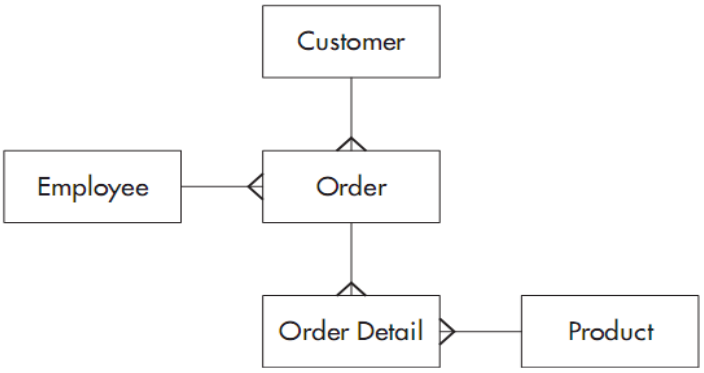

Figure 1-4: Relational model structure

In Figure 1-5, three of the five tables have been represented with sample data in selected columns.

Customer Table

Customer ID	Company Name	Contact First Name	Contact Last Name	Job Title	City	State
6	Company F	Francisco	Pérez-Olaeta	Purchasing Manager	Milwaukee	WI
26	Company Z	Run	Liu	Accounting Assistant	Miami	FL

Order Table

Order ID	Customer ID	Employee ID	Order Date	Shipped Date	Shipping Fee
51	26	9	4/5/2006	4/5/2006	$60.00
56	6	2	4/3/2006	4/3/2006	$ 0.00
79	6	2	6/23/2006	6/23/2006	$ 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 1-5: Relational table contents

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.

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. You might also see variables referred to as *instance variables* or *properties*, but I will stick with the term *variables* for consistency.

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 can be accessed *only* through methods. This property is called *encapsulation*.

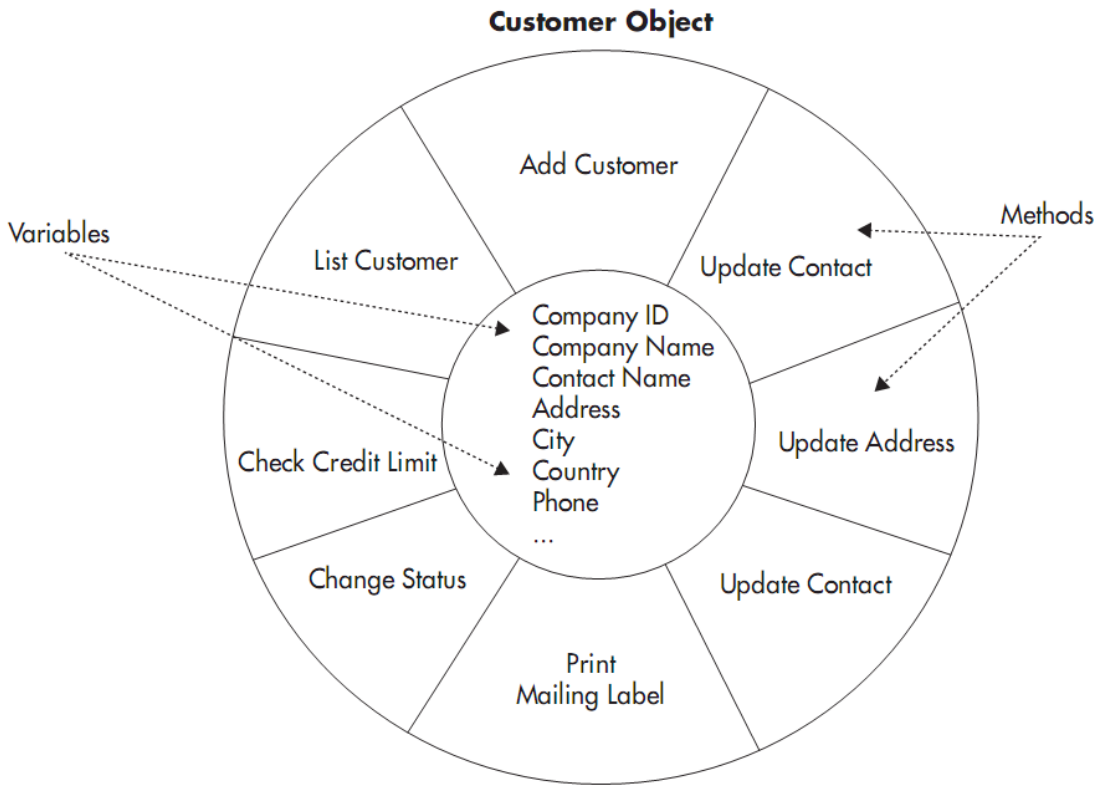

Figure 1-6: The anatomy of an object