CHAPTER 1

Introduction to Database Systems

Welcome and congratulations on your entry to this course in database systems. The fact that you are in this course means that you have covered several fundamental topics in programming, data structures, user interface, and software engineering. Now you want to learn about databases—their significance, the underlying theoretical principles that govern them, how they are constructed, and their management. You are at the right place. This chapter addresses the first issue: the significance of database systems. The sections in this chapter are the following:

- Definition and Rationale
- Objectives of a Database System
- Advantages of a Database System
- Approaches to Database Design
- Desirable Features of a Database System
- Database Development Life Cycle
- Summary and Concluding Remarks

1.1 Definitions and Rationale

A *database system* (DBS) is a computerized record keeping system with the overall purpose of maintaining information and making it available whenever required. The database typically stores related data in a computer system.

A *database management system* (DBMS) is a set of programs that allow for the management of a database. Starting in Chapter 2 and extending to subsequent chapters, we will cover several of the critical functions of a DBMS. Some of the more obvious ones are the following:

- Data definition, which is the creation and management of relations, dependencies, integrity constraints, views, etc.
- Data manipulation, which includes insertion, update, deletion, retrieval, reorganization, and aggregation of data
- System and data security, which means controlling access to the system, resources, and data
- Programming language support

E. C. Foster and S. Godbole, *Database Systems*, DOI 10.1007/978-1-4842-1191-5_1

The main components of a DBS include

- Hardware and operating system
- DBMS
- Database
- Related software systems and/or applications
- Users, including technical users and end users

Database users communicate with the software systems/applications, which in turn communicate (through the programming interface) with the DBMS. The DBMS communicates with the operating system (which in turn communicates with the hardware) to store data in and/or fetch data from the database. Figure 1-1 illustrates these basic concepts.

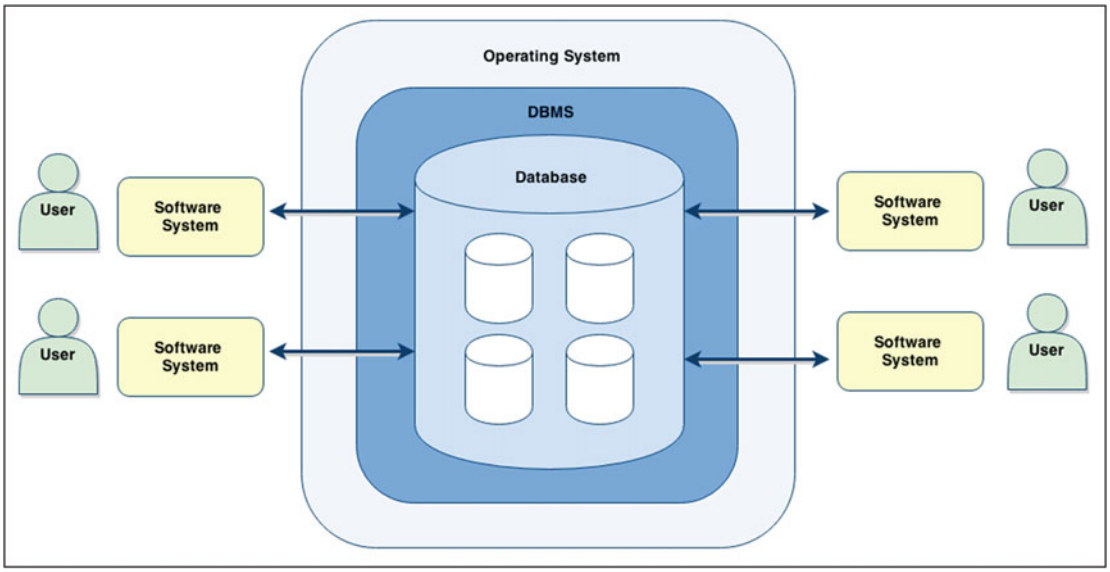

Figure 1-1. *Simplified representation of a DBS*

Databases are essential to software engineering; many software systems have underlying databases that are constantly accessed, though in a manner that is transparent to the end user. Table 1-1 provides some examples. Companies that compete in the marketplace need databases to store and manage their mission critical and other essential data.

Table 1-1. *Illustrations of the Importance of Databases*

Software Category	Database Need
Operating Systems	A sophisticated internal database is needed to keep track of various resources of the computer system including external memory locations, internal memory locations, free space management, system files, user files, etc. These resources are accessed and manipulated by active jobs. A process (also called a job) is created when a user logs on to the system, and is related to the user account. This job can in turn create other jobs, thus creating a job hierarchy. When you consider that in a multi-user environment, there may be several users and hundreds to thousands of jobs, as well as other resources, you should appreciate that underlying an operating system is a very complex database that drives the system.
Compilers	Like an operating system, a compiler has to manage and access a complex dynamic database consisting of syntactic components of a program as it is converted from source code to object code.
Information Systems	Information systems all rely on and manipulate related databases, in order to provide mission critical information for organizations. All categories of information systems are included. Common categories include (but are not confined to) decision support systems (DSS), executive information systems (EIS), management information systems (MIS), web information systems (WIS), enterprise resource planning systems (ERPS), and strategic information systems (SIS).
Expert Systems	At the core of an expert system is a knowledge base containing cognitive data that is accessed and used by an inference engine to draw conclusions based on input fed to the system.
CAD, CAM, and CIM Systems	A computer-aided design (CAD), computer-aided manufacturing (CAM), or computer-integrated manufacturing (CIM) system typically relies on a centralized database (repository) that stores data that is essential to the successful operation of the system.
Desktop Applications	All desktop applications (including hypermedia systems and graphics software) rely on resource databases that provide the facilities that are made available to the user. For example, when you choose to insert a bullet or some other enhancement in a MS Word document, you select this feature from a database containing these features.
CASE and RAD Tools	Like desktop applications, computer-aided software engineering (CASE) tools and rapid application development (RAD) tools rely on complex resource databases to service the user requests and provide the features used.
DBMS Suites	Like CASE and RAD tools, a DBMS also relies on a complex resource databases to service the user requests and provide the features used. Additionally, a DBMS maintains a very sophisticated meta database (called a data dictionary or system catalog) for each user database that is created and managed via the DBMS.

In this course, you will learn how to design, implement, and manage databases. In so doing, you will be exposed to various database technologies and methodologies that are common in the software engineering industry.

Before proceeding further, it is important to make a distinction between *data* and *information*. Data refers to the raw materials that software systems act on in order to produce useful information to end users. Information is processed, and assimilated data that conveys meaning to its intended users. A database is configured to store data. Software systems and/or applications provide the intermediary role of pulling data from the underlying database and synthesizing this data into meaningful information for the end users.

What would life be like without contemporary database systems? If you know someone who is old enough, ask him/her about such an era of filing cabinets, handwritten records, or typewriter-generated documents. Life was very slow then, but it was the norm. Try to fit that lifestyle into 21st century life, and you would have a perfect euphemistic definition of misery. Quite simply, it would not work. Databases are here to stay!

1.2 Objectives of a Database System

There are several primary and secondary objectives of a database system that should concern the computer science (CS) professional. Whether you are planning to design, construct, develop, and implement a DBS, or you are simply shopping around for a DBMS, these objectives help you to develop an early appreciation for the field; they should also provide useful insight into where the course is heading. As you will soon see, these objectives are lofty, and it is by no means easy to achieve them all.

1.2.1 Primary and Secondary Objectives

The primary objectives of a database system include the following:

- Security and protection, which includes the prevention of unauthorized users and protection from inter-process interference
- Reliability, which is the assurance of stable, predictable performance
- Facilitation of multiple users
- Flexibility, including the ability to obtain data and effect action via various methods
- Ease of data access and data change
- Accuracy and consistency
- Clarity, which includes standardization of data to avoid ambiguity
- Ability to service unanticipated requests
- Protection of the investment, typically achieved through backup and recovery procedures
- Minimization of data proliferation, so new application needs may be met with existing data rather than creating new files and programs
- Availability, so that data is available to users whenever it is required

In addition to the above, there are some additional objectives that one may argue are just as important. For want of a better term, let's label these as secondary or additional objectives. Included in these additional objectives are the following:

- Physical Data Independence: Storage hardware and storage techniques are insulated from application programs.
- Logical Data Independence: Data items can be added or subtracted, or the overall logical structure modified, without affecting existing application programs that access the database
- Control of Redundancy: The general rule is to store data minimally and not replicate that storage in multiple places unless this is absolutely necessary.
- Integrity Controls: Range checks and other controls must prevent invalid data from entering the system.
- Clear Data Definition: It is customary to maintain a data dictionary that unambiguously defines each data item stored in the database.
- A Suitably Friendly User Interface: It can be graphical, command-based, or menu-based.
- Tunability: Easy reorganizing the database to improve performance without changing the application programs.
- Automatic Reorganization of Migration: This improves performance.

1.2.2 Clarification on Data Independence

Data independence is an important concept that needs further clarification. Data independence is the immunity of application programs to changes in structure and access strategy of data. It is necessary for the following reasons:

- Different applications and users will need to have different logical views (interpretation) of data.
- The tuning of the system should not affect the application programs.

Physical data independence implies that the user's view is independent of physical file organization, machine, or storage medium. Logical data independence implies that each user (or application program) can have his/her (its) own logical view and does not need a global view of the database.

As an aspiring computer science (CS) professional, you have by now been exposed to various high-level programming languages (HLPLs). These HLPLs have built-in file processing systems that you have no doubt gained mastery in using. You will soon learn that these HLPLs do not support data independence. When you use them to define your data files, there is no separation between the data file you wish to manipulate and the application programs that use them. A database system resolves this dilemma by introducing an additional layer of abstraction between the application programmer and the data files that are manipulated in multiple application programs.

1.3 Advantages of a Database System

A database system brings a number of advantages to its end users as well as the company that owns it. Some of the advantages are the following:

- Redundancy can be reduced.
- Inconsistencies can be avoided.
- Data can be shared.
- Standards can be enforced.
- Security restrictions can be applied.
- Integrity can be maintained.
- Conflicting requirements can be balanced.
- Performance is improved due to speed of processing, reduction in paperwork, etc.
- Maintenance and retrieval of data are very easy—no complicated application program is needed.
- It's not solely dependent on the high level language (HLL) programming for use.
- Logical views of data stored can be easily created.
- Record structures can change without any adverse effect on data retrieval (due to physical and logical data independence).

1.4 Approaches to Database Design

In examining the management of data via computerized systems, it appears that five broad approaches have been pursued over the past few decades:

- Instant small system, which uses one file
- File processing systems, which involve many files
- Other non-relational systems, such as hierarchical, inverted-list, and network approaches
- Relational databases (the focus of this course), which were pioneered by prominent individuals such as Edgar Codd, Ronald Fagin, and Christopher Date, among others
- Object databases, which are an alternate approach (also discussed later in the course)

These five approaches may actually be rearranged into three broad categories, each of which will be clarified. The categories are as follows:

- Conventional files
- Outdated database approaches such as the hierarchical model, network model, and the inverted-list model
- Dominant contemporary database approaches as in the relational model and the object-oriented model

In addition to the dominant contemporary database approaches, there are three emerging database approaches that are worthy of mention (although they will not be fully explored in this course). They are summarized below:

- **Hadoop:** This describes a framework for handling distributed processing of large data sets (see [Apache 2014]
- **Entity-Attributes-Value (EAV) Model:** This approach reduces a database to three principal storage entities: an entity for defining other entities; an entity for defining properties (attributes) of entities; an EAV entity that connects the other two entities and stored values for entity-attribute combinations (see [Wikipedia 2016]).
- **NoSQL:** This approach refers to a family of non-relational database approaches that are designed for managing large data sets, while providing benefits such as flexibility, scalability, availability, lower costs, and special capabilities. Four related methodologies are key-value stores, graph stores, column stores, and document stores (see [IBM 2015]).

1.4.1 Conventional Files

Figure 1-2 illustrates the idea of the conventional file approach. Application programs exist to update files or retrieve information from files. This is a traditional approach to database design that might still abound in very old *legacy systems* (software systems based on old technology and/or methodologies). You have also used this approach in the early stages of your journey as a CS professional. Most HLPLs have built-in file processing systems that you learn to use.

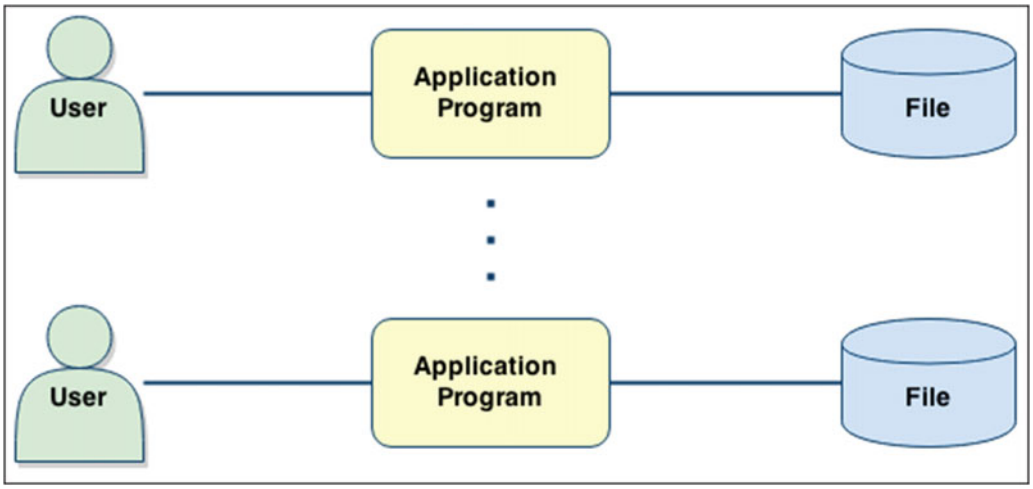

Figure 1-2. *Conventional file-based design*

The main problem with the traditional approach is the absence of data independence. To illustrate the problem, consider for a moment an information system consisting of 30 data files and 150 application programs that manipulate those files. Suppose that each data file impacts 10–15 application programs. Whenever it becomes necessary to adjust the structure of a data file in any way, it will be necessary to track down 10–15 application programs and adjust them as well. Certainly, you do realize this is a very inefficient way of managing a complex software system that may be contingent on a far more complex database.

1.4.2 Database Approach

In the database approach, a database is created and managed via a database management system (DBMS) or CASE tool. A user interface, developed with appropriate application development software, is superimposed on the database, so that end users access the system through the user interface. Figure 1-3 (which is a simplification of Figure 1-1) illustrates the basic idea. All the data resides in the database. Various software systems can then access the database.

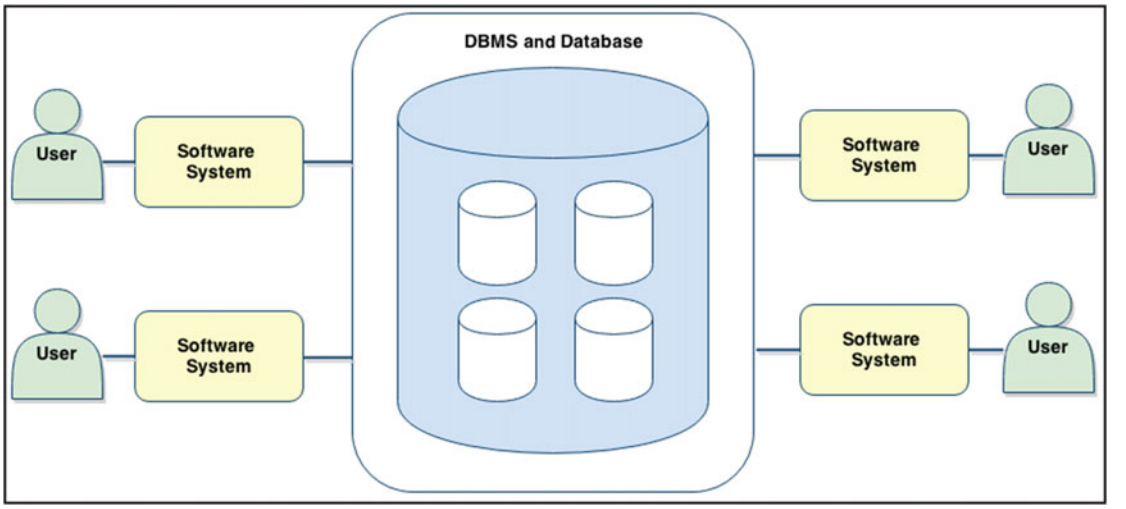

Figure 1-3. *Database approach to data management*

Of the five methodologies for database design, the relational model still dominates contemporary software engineering. Since the 1970s, relational databases have dominated the field of database systems. Object databases created some interest for a while, but it appears that they have been replaced by more contemporary approaches such as the EAV model, Hadoop, and NoSQL. Still, relational databases continue to dominate. Later in the course, you will understand why this dominance is likely to continue. The other three approaches are traditional approaches that have been discarded due to their related problems. They will not be discussed any further; for more information on them, see the recommended readings.

1.5 Desirable Features of a DBS

Contemporary database systems must live up to de facto standards set by the software engineering industry. Roughly speaking, a well-designed database system must exhibit the following features (more specific standards will be discussed later in the course):

- Provide most of the advantages mentioned earlier
- Meet most of the objectives mentioned earlier
- Provide for easy communication with other systems
- Be platform independent
- Be thoroughly documented

- Provide comprehensive backup and recovery features
- Provide efficient and effective programming support
- Offer a comprehensive system catalog
- Offer appropriate transaction management

The importance of these features will become clearer as you proceed through the course. You will also see that for the most part, the choice of the DBMS goes a far way in determining the characteristic features of the database system and how they are provided.

1.6 Database Development Life Cycle

You are perhaps familiar with, or will soon learn about, the software development life cycle (SDLC) from your software engineering course(s). The SDLC, which is summarized in Table 1-2, outlines the various phases that software systems go through. Each phase actually includes multiple steps that are best covered in a software engineering course (for instance, see [Foster 2014] and [Schach 2011]).

Table 1-2. *Software Development Life Cycle*

SDLC Phase	Related Deliverable(s)
Investigation and Analysis	Initial System Requirements; Requirements Specification
Design (Modeling)	Design Specification
Development (Construction)	Actual Software; Product Documentation
Implementation	Actual Software; Product Documentation
Management	Enhanced Software; Revised Documentation

A database system qualifies as a software system. Moreover, the DBMS (which you typically use to create and manage databases) is one of the most complex software systems that you will encounter and work closely with. As you continue through this course and subsequently work with databases, this will become even clearer to you.

As mentioned earlier (section 1.1), a database does not exist in a vacuum, but is usually part of a software system. A *database development life cycle* (DDLC) may therefore be perceived from two perspectives:

- It may be viewed as being identical and concurrent with the SDLC. At each phase in the SDLC, consideration is given to the database as an integral part of the software product.
- If we consider that in many cases, the database has to be constructed and implemented, and managed as a separate resource that various software systems can tap into, then we may construct a similar but different life cycle for the database as illustrated in Table 1-3.

Table 1-3. *Database Development Life Cycle*

DDLC Phase	Related Deliverable(s)
Database Investigation and Analysis	Initial Database Requirements
Database Modeling	Database Model
Database Designing	Database Design Specification
Database Development	Actual Database
Implementation	Actual Database in Use
Management	Enhanced Database; Revised Database Documentation

If you compare Tables 1-2 and 1-3, you will see clearly that a database system is really a specialized software system. Here are a few additional points to remember:

- Applying basic investigation strategies and methodologies that are covered in your software engineering course, you will be able to navigate the database investigation and analysis phase. Appendix 3 provides a summary of these strategies and methodologies. This course primarily concentrates on the other phases.
- With experience, the database modeling and database designing phases can be merged into one phase. This will be further clarified in Chapters 3 through 5. However, as a new learner of database systems, you should not rush into this. It is strongly advised that you keep them separate! After a few years of practice, you should be able to look back and smile at the concepts you once struggled with.
- Once the database is in the implementation phase, management of it becomes an ongoing experience, until the database becomes irrelevant to the organization.

1.7 Summary and Concluding Remarks

Let's summarize what we have covered in this chapter:

- A database system is a computerized record keeping system with the overall purpose of maintaining information and making it available on demand.
- The DBMS is the software that facilitates creation and administration of the database.
- The DBS is made up of the hardware, the operating system, the DBMS, the actual database, the application programs, and the end users.
- There are several primary and secondary objectives of a DBS, which are of importance to the CS professional.
- Many software systems rely on underlying database systems to provide critical information.
- A DBS brings a number of significant advantages to the business environment.
- There are three traditional approaches to constructing a DBS that are no longer prevalent today. They are the instant small system, the file processing system, and the traditional non-relational approaches.

- There are five contemporary approaches to constructing a DBS. They are the relational approach, the object-oriented approach, the Hadoop framework, the EAV approach, and the NoSQL approach. The relational approach is the most dominant.
- In striving to acquire a DBS, it is advisable to aspire for most of the objectives and advantages. Additionally, one should aim for user friendliness, thorough documentation, and a DBMS that provides platform independence, comprehensive system catalog, backup and recovery, appropriate transaction management, communication with other systems, and adequate programming support.
- The database development life cycle outlines the main activities in the useful life of a DBS.

Interested? We have just begun to touch the surface. There is a lot more to cover. Most successful software systems are characterized by carefully designed databases. In fact, it is safe to say that the efficacy of the software system is a function of its underlying database. So stay tuned: the next chapter provides more clarification on the database environment.

1.8 Review Questions

Here are some review questions for you to answer. You are encouraged to write your responses down; that way you will know whether you need to revisit related sections of the chapter.

1. What is a database system?
2. Why are database systems important?
3. What is a database management system?
4. What are the objectives (primary and secondary) of a DBS?
5. What is data independence, and how important is it?
6. What are the advantages of a DBS?
7. What are the possible approaches to acquiring a DBS?
8. How do database systems relate to software engineering?
9. Compare the software development life cycle to the database development life cycle.

1.9 References and/or Recommended Readings

[Apache 2014] Apace Software Foundation. 2014. “Hadoop.” Accessed February 2016. http://hadoop.apache.org/.

[Connolly, 2015] Connolly, Thomas and Carolyn Begg. 2015. *Database Systems: A Practical Approach to Design, Implementation and Management* 6th ed. Boston: Pearson. See Chapter 1.

[Coronel, 2015] Coronel, Carlos and Steven Morris. 2015. *Database Systems: Design, Implementation and Management* 11th ed. Boston: Cengage Learning. See Chapter 1.

[Date, 2004] Date, Christopher J. 2004. *Introduction to Database Systems* 8h ed. Menlo Park, CA: Addison-Wesley. See Chapter 1.

[Elmasri, 2011] Elmasri, Ramez and Shamkant B. Navathe. 2011. *Fundamentals of Database Systems* 6th ed. Boston: Pearson. See Chapter 1.

[Foster 2014] Foster, Elvis C. *Software Engineering — a Methodical Approach*. New York: Apress Publishing.

[Garcia-Molina, 2009] Garcia-Molina, Hector, Jeffrey Ullman and Jennifer Widom. 2009. *Database Systems: The Complete Book* 2nd ed. Boston: Pearson. See Chapter 1.

[Hoffer 2013] Hoffer, Jeffrey A., Ramesh Venkataraman, and Heikki Topi. 2013. *Modern Database Management* 11th ed. Boston: Pearson. See Chapter 1.

[IBM 2015] IBM Corporation. 2015. "Analytics White Paper." Accessed February, 2016. https://cloudant.com/wp-content/uploads/Why_NoSQL_IBM_Cloudant.pdf.

[Kifer 2006] Kifer, Michael, Arthur Bernstein, and Phillip M. Lewis. 2006. *Databases and Transaction Processing: An Application Oriented Approach* 2nd ed. Boston: Pearson. See Chapters 1 and 2.

[Pratt 2015] Pratt, Phillip J. and Mary Z. Last. 2015. *Concepts of Database Management* 8th ed. Boston: Course Technology. See Chapter 1.

[Schach 2011] Schach, Stephen R. 2011. *Object-Oriented and Classical Software Engineering* 8th ed. Boston: McGraw-Hill.

[Ullman 2008] Ullman, Jeffrey D., and Jennifer Widom. 2008. *A First Course in Database Systems* 3rd ed. Boston: Pearson. See Chapter 1.

[Wikipedia 2016] Wikipedia. 2016. "Entity-Attribute-Value Model." Accessed February 2016. https://en.wikipedia.org/wiki/Entity%E2%80%93attribute%E2%80%93value_model.