

1

Fundamental Concepts of Database Management



Chapter Objectives

In this chapter, you will learn to:

- understand the differences between the file versus database approach to data management;
- discern the key elements of a database system;
- identify the advantages of database systems and database management.

Opening Scenario

Since Sober is a startup company, it must carefully decide how it will manage all its data. The company is thinking about storing all its data in Word documents, Excel files, and maybe some other files (e.g., Notepad) as well.

In this chapter, we discuss the fundamental concepts of database management. Many ideas presented here are elaborated in later chapters. We kick off by reviewing popular applications of database technology, and follow this by defining key concepts such as a database and a database management system, or DBMS. Next, we step back in time and discuss the file-based approach and contrast it with the database approach to data management. We then zoom into the elements of a database system. We conclude by discussing the advantages of database design.

1.1 Applications of Database Technology

Data are everywhere and come in different shapes and volumes. These data need to be stored and managed using appropriate data management or database technologies. Think about the storage and retrieval of traditional numeric and alphanumeric data in an application developed to keep track of the number of products in stock. For each product, the product number, product name, and available quantity needs to be stored. Replenishment orders need to be issued as soon as the quantity drops below the safety limit. Every replenishment order has an order number, order date, supplier number, supplier name, and a set of product numbers, names, and quantities.

Database technology is not just for traditional numeric and alphanumeric data. It can also store multimedia data such as pictures, audio, or video – YouTube and Spotify support the querying of music based upon artist, album, genre, playlist, or record label. Biometric data, including fingerprints and retina scans, are often used for security, such as border control as you enter a country. Information is also gathered by wearables, such as a Fitbit or an Apple Watch, which continuously monitor and analyze your health and fitness. *Geographical information systems (GIS)* applications, such as Google Maps, store and retrieve all types of spatial or geographical data.

Database technology can also store and retrieve *volatile* data. One example is high-frequency trading, where automated, algorithmic platforms are used by investment banks or hedge funds to process a large number of orders at extremely high speed based upon events happening in the environment or macro-economy. Another example is sensors monitoring the key parameters of a

nuclear reactor, whereby an automatic system shutdown may be enacted if certain thresholds are hit.

You may have heard the term *Big Data*, referring to the huge amounts of data being gathered and analyzed by companies such as Google, Facebook, and Twitter. Look at Walmart, America's largest retailer with over 11,000 locations worldwide, \$4.8 billion in annual sales and over 100 million customers per week. Its point-of-sale (POS) database system stores an enormous amount of data such as which customer bought what products, in what quantities, at which location, and at what time. All these data can then be intelligently analyzed using analytical data modeling to reveal unknown but interesting purchase patterns, such as which products are frequently purchased together. Better still, certain analysis techniques allow one to make predictions about the future (e.g., which customers are most likely to respond positively to a sales promotion). We discuss this in more detail in [Chapter 20](#).

These are just a few examples of database applications; many others exist.

Drill Down

The Internet of Things (IoT) provides many examples of Big Data applications. Moocall is a Dublin-based startup providing sensors for farmers to reduce the mortality rates of calves and cows during birthing. The sensor is attached to the cow's tail. They measure specific movements of the tail triggered by labor contractions as the calving begins. These sensor data are then sent through the Vodafone IoT network to a farmer's smartphone. Using an app, the farmer gets up-to-date information about the calving process and can intervene or call a vet when needed. The app can generate alerts, and includes a herd management facility. This technology improves both the farmer's

productivity and the survival probabilities of calves and cows during the birthing process.

Retention Questions

- Give some examples of applications of database technology.

1.2 Key Definitions

We have briefly introduced the concept of a database by exploring the various types of databases you may encounter every day. A [database](#) can be defined as a collection of related data items within a specific business process or problem setting. Consider a purchase order system, where you have data items such as products, suppliers, and purchase orders. Each data item has characteristics: a product has a product number, product name, and product color; a supplier has a supplier name and a supplier address; a purchase order has a reference number and date. These data items are also related. A product can be supplied by one or more suppliers. A purchase order is always connected to exactly one supplier. A supplier can supply one or more products. These are examples of relationships between the data items that should be adequately captured by a database. A database has a target group of users and applications. An inventory manager uses our purchase order system to manage the inventory and issue purchase orders; a product manager uses it for monitoring trends in product sales.

A [database management system \(DBMS\)](#) is the software package used to define, create, use, and maintain a database. It typically consists of several software modules, each with their own functionality, as we discuss in [Chapter 2](#). Popular DBMS vendors are Oracle, Microsoft, and IBM. MySQL is a well-known open-source DBMS. The combination of a DBMS and a database is then often called a [database system](#).

Drill Down

Gartner¹ estimated the total DBMS market value at \$35.9 billion for 2015, which represented an 8.7% growth when compared to 2014. According to the IDC, the overall market for database management solutions is estimated to reach over \$50 billion by 2018.

Connections

In [Chapter 2](#) we discuss the internal architecture of a DBMS. We also provide a categorization of DBMSs along various dimensions.

Retention Questions

- Define the following concepts:
 - database
 - DBMS
 - database system

1.3 File versus Database Approach to Data Management

Before we further explore database technology, let's step back and see how data management has evolved. This will give us a proper understanding of the legacy problems many companies are still facing.

1.3.1 The File-Based Approach

In the early days of computing, every application stored its data into its own dedicated files. This is known as a [file-based approach](#) and is illustrated in [Figure 1.1](#).

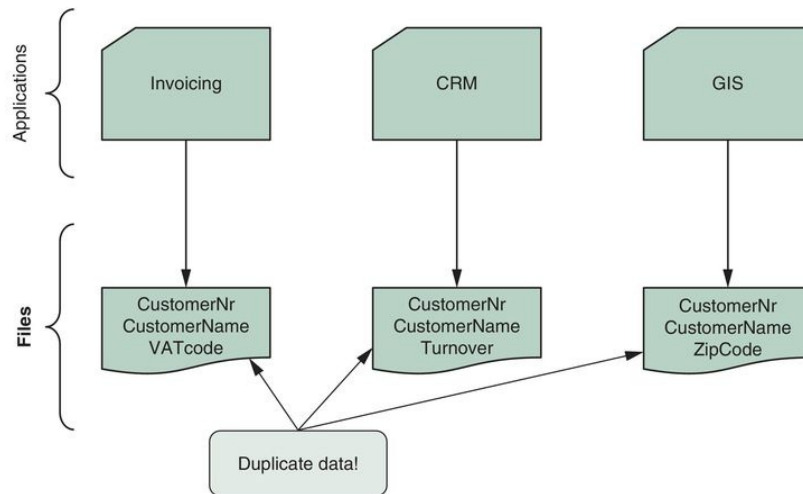


Figure 1.1 File-based approach to data management.

Suppose we have a traditional invoicing application, written in a programming language such as COBOL or C, that makes use of customer information such as customer number, customer name, VAT code, etc., stored in a separate file. A separate application, such as a customer relationship management (CRM) system, makes use of a different file containing the same data. Finally, a third application (GIS) stores information such as customer number, customer name, and ZIP code in yet another file. The data files only contain the data themselves; the data definitions and descriptions are included in each application separately. An application can make use of one or more files. As more applications are developed with corresponding data files, this file-based approach to data management will cause serious problems.

Since each application uses its own data files and many applications use similar data, duplicate or redundant information will be stored, which is a waste of storage resources. If this is not appropriately managed there is a danger that customer data will be updated in only one file and not elsewhere, resulting in inconsistent data. In this file-based approach to data management there is a strong coupling, or dependency, between the applications and the data. A structural change in a data file necessitates changes in all applications that use it, which is not desirable from a maintenance perspective. It is hard to manage *concurrency control* (i.e., the simultaneous access by different users or applications to the same data without conflicts). For example, if one application performs a cash transfer while another application calculates the account balance, and the data operations of both applications are interleaved for efficiency, this can easily lead to inconsistent data in cases where there are no adequate concurrency control facilities provided. Since the applications each work independently with their own ecosystem of data files, it is difficult and expensive to integrate applications aimed at providing cross-company services. Although this file approach to data management has serious disadvantages, many firms still struggle with “*legacy*” file-based systems in their current information and communications technology (ICT) environment.

1.3.2 The Database Approach

The emergence of database technology provided a new paradigm for data management. In this [database approach](#), all data are stored and managed centrally by a DBMS, as illustrated in [Figure 1.2](#).

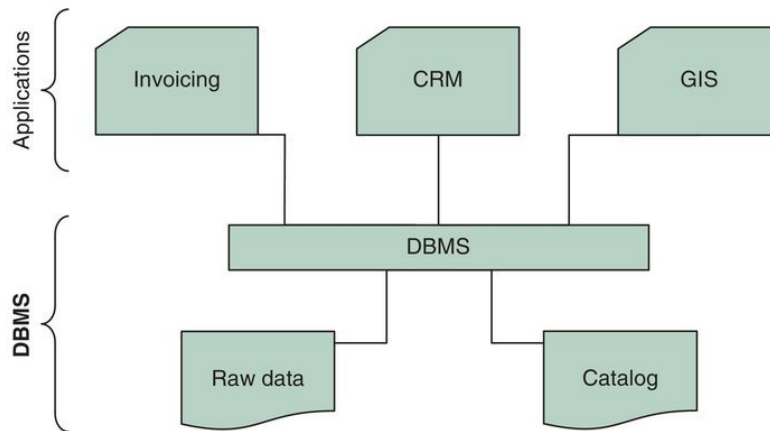


Figure 1.2 Database approach to data management.

The applications now directly interface with the DBMS instead of with their own files. The DBMS delivers the desired data at the request of each application. The DBMS stores and manages two types of data: raw data and metadata. [Metadata](#) refers to the data definitions that are now stored in the catalog of the DBMS. This is a key difference to the file-based approach. The metadata are no longer included in the applications, but are now properly managed by the DBMS itself. From an efficiency, consistency, and maintenance perspective, this approach is superior.

Another key advantage of the database approach is the facilities provided for data querying and retrieval. In the file-based approach, every application had to explicitly write its own query and access procedures. Consider the following example in pseudo-code:

```
Procedure FindCustomer;  
Begin  
    open file Customer.txt;  
    Read(Customer)  
    While not EOF(Customer)  
        If Customer.name='Bart' Then  
            display(Customer);  
        EndIf  
    Read(Customer);  
    EndWhile;  
End;
```

Here, we first open a Customer.txt file and read the first record. We then implement a while loop that iterates through each record in the file until the end of the file is reached (indicated by EOF(Customer)). If the desired information is found (Customer.name='Bart'), it will be displayed. This requires a lot of coding. Because of the tight coupling between data and applications, many procedures would be repeated in various applications, which is again not very appealing from a maintenance perspective. As noted, DBMSs provide database languages that facilitate both data querying and access. A well-known language, which we discuss extensively in [Chapter 7](#), is Structured Query Language (SQL). SQL can be used to formulate database queries in a structured and user-friendly way, and is one of the most popular data querying standards used in the industry. An example SQL query that gives the same output as our pseudo-code above could be:

```
SELECT *  
FROM Customer  
WHERE  
name = 'Bart'
```

Here, you only need to specify what information you want. In our case, we want all customer information for customer 'Bart'. This SQL query will then be executed by the DBMS in a transparent way. In the database approach, we only need to specify which data we are interested in, and no longer how we should access and retrieve them. This facilitates the development of database applications because we no longer need to write complex data retrieval procedures.

To summarize, the file-based approach results in a strong application–data dependence, whereas the database approach allows for applications to be independent from the data and data definitions.

Drill Down

One of the key disadvantages of a file-based approach to data management is that the data typically sit all over the organization in silos; therefore, an overall, comprehensive view is lacking. For example, the city of Amsterdam has data spread across 12,000 different datasets. Because of the lack of integration, no one knows exactly how many bridges span Amsterdam's famous canals, because each of the city's individual districts has its own data and no overall comprehensive database is available. It turned out that many of these siloed datasets adopted their own data definition of a bridge, which further complicates matters. See <http://sloanreview.mit.edu/case-study/lessons-from-becoming-a-data-driven-organization>.

Retention Questions

- Contrast the file versus database approach to data management.

1.4 Elements of a Database System

In this section we discuss database model versus instances, data models, the three-layer architecture, the role of the catalog, the various types of database users, and DBMS languages.

1.4.1 Database Model versus Instances

In any database implementation, it is important to distinguish between the description of the data, or data definitions, and the actual data. The [database model](#) or [database schema](#) provides the description of the database data at different levels of detail and specifies the various data items, their characteristics, and relationships, constraints, storage details, etc.² The database model is specified during database design and is not expected to change frequently. It is stored in the catalog, which is the heart of the DBMS. The [database state](#) then represents the data in the database at a particular moment. It is sometimes also called the *current set of instances*. Depending upon data manipulations, such as adding, updating, or removing data, it typically changes on an ongoing basis.

The following are examples of data definitions that are an essential part of the database model stored in the catalog.

Database model

Student (number, name, address, email)

Course (number, name)

Building (number, address)

...

We have three data items: Student, Course, and Building. Each of these data items can be described in terms of its characteristics. A student is characterized by a number, name, address, and email; a course by a number and name; and a building by a number and address.

[Figure 1.3](#) shows an example of a corresponding database state. You can see the database includes data about three students, three courses, and three buildings.

STUDENT			
Number	Name	Address	Email
0165854	Bart Baesens	1040 Market Street, SF	Bart.Baesens@kuleuven.be
0168975	Seppe vanden Broucke	520, Fifth Avenue, NY	Seppe.vandenbroucke@kuleuven.be
0157895	Wilfried Lemahieu	644, Wacker Drive, Chicago	Wilfried.Lemahieu@kuleuven.be

COURSE	
Number	Name
D0I69A	Principles of Database Management
D0R04A	Basic Programming
D0T21A	Big Data & Analytics

BUILDING	
Number	Address
0600	Naamsestraat 69, Leuven
0365	Naamsestraat 78, Leuven
0589	Tiensestraat 115, Leuven

Figure 1.3 Example database state.

1.4.2 Data Model

A database model comprises different data models, each describing the data from different perspectives. A good data model is the start of every successful database application. It provides a clear and unambiguous description of the data items, their relationships, and various data constraints from a particular perspective. Several types of data models will be developed during a database design process.

A [conceptual data model](#) provides a high-level description of the data items (e.g., supplier, product) with their characteristics (e.g., supplier name, product number) and relationships (e.g., a supplier can supply products). It is a communication instrument between the information architect (see [Chapter 4](#)) and business user to make sure the data requirements are adequately captured and modeled. Therefore, the conceptual data model should be implementation-independent, user-friendly, and close to how the business user perceives the data. It will usually be represented using an Enhanced Entity Relationship (EER) model or an object-oriented model, as we discuss in [Chapter 3](#).

A [logical data model](#) is a translation or mapping of the conceptual data model toward a specific implementation environment. The logical data items may still be understood by business users, but are not too far removed from the physical data organization. Depending upon the ICT environment available, it can be a hierarchical (see [Chapter 5](#)), CODASYL (see [Chapter 5](#)), relational (see [Chapters 6 and 7](#)), object-oriented (see [Chapter 8](#)), extended relational (see [Chapter 9](#)), XML (see [Chapter 10](#)), or NoSQL model (see [Chapter 11](#)).

The logical data model can be mapped to an internal data model that represents the data's physical storage details. It clearly describes which data are stored where, in what format, which indexes are provided to speed up retrieval,

etc. It is therefore highly DBMS-specific. We discuss internal data models in [Chapters 12](#) and [13](#).

The **[external data model](#)** contains various subsets of the data items in the logical model, also called views, tailored toward the needs of specific applications or groups of users.

Connections

In [Chapter 3](#) we discuss the EER and UML conceptual data models in more detail. Later chapters cover logical (and sometimes external) data models: the hierarchical and CODASYL model in [Chapter 5](#), the relational model in [Chapters 6](#) and [7](#), the object-oriented model in [Chapter 8](#), the extended relational model in [Chapter 9](#), the XML data model in [Chapter 10](#) and various NoSQL data models in [Chapter 11](#). [Chapters 12](#) and [13](#) elaborate on internal data models.

1.4.3 The Three-Layer Architecture

The [three-layer architecture](#) is an essential element of every database application and describes how the different underlying data models are related.³ It is illustrated in [Figure 1.4](#).

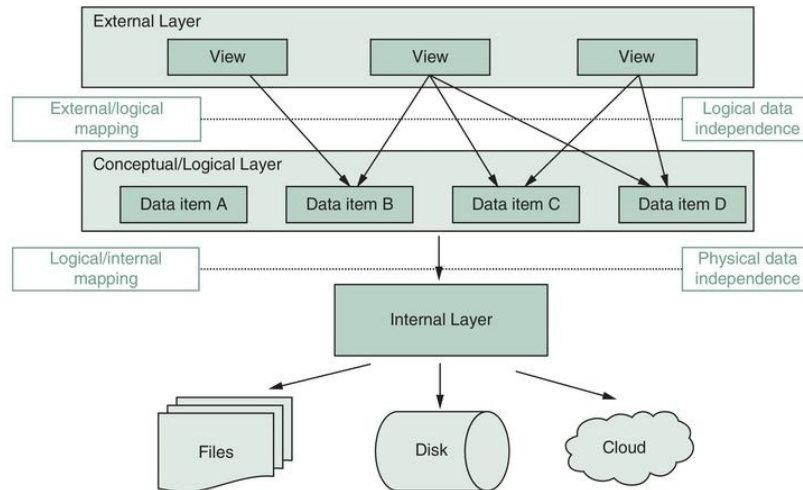


Figure 1.4 The three-layer database architecture.

We start with the conceptual/logical layer. Here, we have the conceptual and logical data models. Both focus on the data items, their characteristics, and relationships without bothering too much about the actual physical DBMS implementation. The conceptual data model should be a user-friendly, implementation-independent, and transparent data model, constructed in close collaboration between the information architect and business user(s). It will be refined to a logical data model based upon the implementation environment.

In the external layer we have the external data model, which includes views offering a window on a carefully selected part of the logical data model. A [view](#) describes the part of the database that a particular application or user group is interested in, hiding the rest of the database. It is used to control data access and enforce security. The views will be tailored to the data needs of an application or

(group of) user(s). A view can serve one or more applications. Consider a view offering only student information to a student registration application, or a view offering only building information to a capacity planning application.

The [internal layer](#) includes the [internal data model](#), which specifies how the data are stored or organized physically. Ideally, changes in one layer should have no to minimal impact on the others. It should be possible to physically reorganize the data with little impact on the conceptual/logical or external layer (physical data independence). Likewise, changes to the conceptual/logical layer can be made with minimal impact on the external layer (logical data independence). We elaborate on both types of data independence in [Section 1.5.1](#).

[Figure 1.5](#) illustrates the three-layer architecture for a procurement business process. The conceptual/logical layer defines the data items such as Product, Customer, Invoice, and Delivery. The internal layer contains the physical storage details specifying how and where the data are stored. The external layer has three views offering specific information to the finance, customer service, and logistics departments. This three-layer database architecture has several advantages in efficiency, maintenance, performance, security, etc.

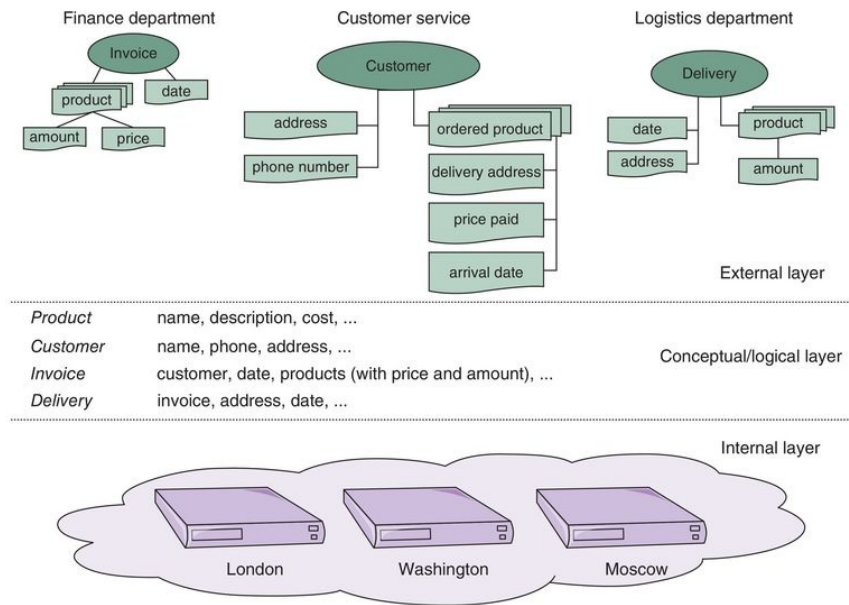


Figure 1.5 Three-layer database architecture for a business procurement process.

1.4.4 Catalog

The [catalog](#) is the heart of the DBMS. It contains the data definitions, or metadata, of your database application. It stores the definitions of the views, logical and internal data models, and synchronizes these three data models to ensure their consistency.⁴

1.4.5 Database Users

As we discuss more extensively in [Chapter 4](#), various types of users interact with the database. An *information architect* designs the conceptual data model. He/she closely interacts with the *business user* to make sure the data requirements are fully understood and modeled. A database designer translates the conceptual data model into a logical and internal data model. The *database administrator* (DBA) is responsible for the implementation and monitoring of the database. He/she sets up the database infrastructure and continuously monitors its performance by inspecting key performance indicators such as response times, throughput rates, and storage space consumed (see [Section 1.5.9](#)). The *application developer* develops database applications in a general-purpose programming language such as Java or Python. He/she provides the data requirements, which are then translated by the database designer or DBA into view definitions. The business user will run these applications to perform specific database operations. He/she can also directly query the database using interactive querying facilities for reporting purposes.

1.4.6 Database Languages

Every DBMS comes with one or more accompanying database languages. The [data definition language \(DDL\)](#) is used by the DBA to express the database's external, logical, and internal data models. These definitions are stored in the catalog. The [data manipulation language \(DML\)](#) is used to retrieve, insert, delete, and modify data. DML statements can be embedded in a general-purpose programming language, or entered interactively through a front-end querying tool. SQL offers both DDL and DML statements for relational database systems (see [Chapter 7](#)).

Retention Questions

- What are the key elements of a database system?
- Discuss the three-layer architecture of a database application. Illustrate with an example.
- What is a catalog and why is it needed?

1.5 Advantages of Database Systems and Database Management

Databases, if adequately designed and managed, offer advantages such as data independence; managing structured, semi-structured, and unstructured data; database modeling; managing data redundancy; specifying integrity rules; concurrency control; backup and recovery facilities; data security and performance utilities. We elaborate on these elements in this section.

1.5.1 Data Independence

Data independence means changes in data definitions have minimal to no impact on the applications using the data. These changes may occur in the internal or the conceptual/logical layer. Physical data independence implies that neither the applications, views, or logical data model must be changed when changes are made to the data storage specifications in the internal data model. Consider reorganizing the data across different storage locations or media, the definition of new access paths or indexes, etc. The applications will keep running successfully, and may be even faster than they were before because of the physical reorganization of the data. To adequately guarantee physical data independence, the DBMS should provide interfaces between the logical and internal data models.

Logical data independence implies that software applications are minimally affected by changes in the conceptual or logical data model. Consider the example of adding new data items, characteristics, or relationships. The views in the external data model will act as a protective shield and mitigate the effect of these modifications on the applications. To guarantee logical data independence, the DBMS must provide interfaces between the conceptual/logical and external layer.

1.5.2 Database Modeling

A data model is an explicit representation of the data items together with their characteristics and relationships. It can also include integrity rules and functions. A conceptual data model should provide a formal and perfect mapping of the data requirements of the business process and is made in close collaboration with the business user. It is then translated into a logical data model and, finally, an internal data model. Unfortunately, a best-case scenario with perfect mapping is often unrealistic, and it is important that a data model's assumptions and shortcomings are clearly documented. Popular examples of data models are the hierarchical model, the CODASYL model, the (E)ER model, the relational model, and the object-oriented model. We discuss these more extensively in [Chapters 5–8](#).

1.5.3 Managing Structured, Semi-Structured, and Unstructured Data

It is important to note that not all kinds of data can be described according to a formal logical data model. This is only possible for [structured data](#), which was the only kind of data the earlier DBMS implementations focused on. With structured data, individual characteristics of data items can be identified and formally specified, such as the number, name, address, and email of a student, or the number and name of a course. The advantage is the ability to express integrity rules and in this way enforce the correctness of the data. As we will discuss in, e.g., [Chapters 7–9](#), it also facilitates searching, processing, and analyzing data, because both the DBMS and the data processing applications have fine-grain control over the data. They can, for example, discriminate between a series of characters representing a student's name and a student's address. In this way, it becomes possible to retrieve, for example, all the names of students that live in New York.

With [unstructured data](#), there are no finer-grain components in a file or series of characters that can be interpreted in a meaningful way by a DBMS or application. Consider a long text document containing the biographies of famous New York citizens. In this plain text it is possible to search for the terms “name”, “student”, and “New York” occurring closely together, but it is impossible to assess whether they pertain to students who lived in New York, students who were born in New York or maybe even students for which the text explains they always wore the same sweater, with the imprint “New York” on it. Moreover, it is not possible to retrieve only the series of characters that represent these students' names. In spite of that, many recent database management systems provide facilities to efficiently store and search such full-text documents. This is especially important, since the volume of unstructured data largely surpasses that

of structured data in most organizations. These unstructured data may contain lots of useful information, if they can be extracted efficiently. Consider improving customer interaction by storing and analyzing complaints letters, classifying legal documents according to their content, or assessing the market's sentiment toward a new product by analyzing tweets that refer to the product. Moreover, modern-day DBMSs are not restricted to storing and managing unstructured textual data, but other kinds of data as well, such as still images, video, and audio.

Finally, it should be stressed that not all data are completely structured or completely unstructured. In later chapters we will discuss how recent DBMS types, such as XML databases ([Chapter 10](#)) and NoSQL databases ([Chapter 11](#)), aim explicitly at dealing efficiently with **semi-structured data**. These are data that have a certain structure, but the structure may be very irregular or highly volatile. Typical examples are individual users' webpages on a large social media platform, or résumé documents in a human resources database, which may loosely exhibit the same structure, but which do not comply entirely with a single, rigid format.

1.5.4 Managing Data Redundancy

One of the key drawbacks of the file-based approach to data management is undesirable duplication of data, which can easily lead to inconsistent data. In the database approach, redundant data can be successfully managed. Duplication of data can be desirable in distributed environments to improve data retrieval performance by providing local access to data rather than using resource-intensive network connections. The DBMS is now responsible for the management of the redundancy by providing synchronization facilities to safeguard data consistency. As an example, an update of a local data copy will be automatically propagated to all duplicate data copies stored at other locations. Compared to the file approach, the DBMS guarantees correctness of the data. It also requires no user intervention and is much more efficient and less error-prone.

1.5.5 Specifying Integrity Rules

Data integrity rules can also be explicitly defined. These rules can be used to enforce the correctness of the data. Syntactical rules specify how the data should be represented and stored. Examples are: customerID should be represented as an integer (e.g., 100, 125, and 200 are correct, but 1.20 or 2a are not); birth date should be stored as month, day, and year (e.g., 02/27/1975 is correct, but 27/02/1975 is not). Semantic rules focus on the semantic correctness or meaning of the data. Examples are: customerID should be unique; account balance should be bigger than 0; and a customer cannot be deleted if he/she has pending invoices. In the file-based approach, these integrity rules have to be embedded in every single application. In the database approach, they are specified as part of the conceptual/logical data model and are stored centrally in the catalog. This substantially improves the efficiency and maintainability of the applications since the integrity rules are now directly enforced by the DBMS whenever anything is updated. In the file-based approach, the applications themselves have to explicitly manage all integrity rules, resulting into a lot of duplication of code, with the accompanying risk of inconsistencies.

1.5.6 Concurrency Control

A DBMS has built-in facilities to support concurrent or parallel execution of database programs, which allows for good performance. A key concept is a database transaction that is a sequence of read/write operations, considered to be an atomic unit in the sense that either all operations are executed or none at all (more details on transactions are provided in [Chapter 14](#)). Typically, these read/write operations can be executed at the same time by the DBMS. However, this should be carefully supervised to avoid inconsistencies. Let's illustrate this with an example ([Table 1.1](#)).

Table 1.1 Illustrating concurrency control

Time	T1	T2	Balance
t1		Begin transaction	\$100
t2	Begin transaction	read(balance)	\$100
t3	read(balance)	balance = balance + 120	\$100
t4	balance = balance – 50	write(balance)	\$220
t5	write(balance)	End transaction	\$50
t6	End transaction		\$50

[Table 1.1](#) shows two database transactions: T1 and T2. T1 updates the account balance by withdrawing \$50. T2 deposits \$120. The starting balance is \$100. If both transactions were to run sequentially, instead of in parallel, the ending balance should be $\$100 - \$50 + \$120 = \170 . If the DBMS interleaves the actions of both transactions, we get the following. T2 reads the balance at t2 and

finds it is \$100. T1 reads the balance at t3 and finds it is \$100. At t3, T2 updates the balance to \$220. However, it still needs to write (or save) this value. At t4, T1 calculates the balance as $\$100 - \$50 = \$50$ whereas T2 saves the balance, which now becomes \$220. T1 then saves the balance as \$50 at t5. It overwrites the value of \$220 with \$50, after which both transactions are ended. Since T1 updates the balance based on the value it had before the update by T2, and then writes the updated balance after T2 is finished, the update effect of T2 is lost. It is as if transaction T2 did not take place. This is commonly called a lost-update problem. The DBMS should avoid the inconsistencies that emanate from the interference between simultaneous transactions.

To ensure database transactions are processed in a reliable way, the DBMS must support the *ACID* (**A**tomicity, **C**onsistency, **I**solation, **D**urability) properties. *Atomicity*, or the all-or-nothing property, requires that a transaction should either be executed in its entirety or not at all. *Consistency* assures that a transaction brings the database from one consistent state to another. *Isolation* ensures that the effect of concurrent transactions should be the same as if they had been executed in isolation. Finally, *durability* ensures that the database changes made by a transaction declared successful can be made permanent under all circumstances.

1.5.7 Backup and Recovery Facilities

A key advantage of using databases is the availability of backup and recovery facilities. These facilities can be used to deal with the effect of loss of data due to hardware or network errors, or bugs in system or application software. Typically, backup facilities can perform either a full or incremental backup. In the latter case, only the updates since the previous backup will be considered. Recovery facilities allow restoration of data to a previous state after loss or damage.

Connections

[Chapter 14](#) introduces the basics of transactions, transaction management, recovery, and concurrency control. It describes how the interplay between these concepts guarantees concurrent access by different users to shared data. [Chapter 16](#) then further elaborates on this by reviewing distributed transaction management.

1.5.8 Data Security

Data security can be directly enforced by the DBMS. Depending on the business application considered, some users have read access, while others have write access to the data (role-based functionality). This can also be further refined to certain parts of the data. Trends such as e-business, B2B (business-to-business), B2C (business-to-consumer), and CRM stress the importance of data security because they increasingly expose databases to internal and external parties. Consider the example of vendor-managed inventory (VMI), where a company can get access to inventory details of its downstream supply chain partner. Using the right security policies should enforce that only read access is provided and no information from competitor products can be retrieved. Data access can be managed via logins and passwords assigned to users or user accounts. Each account has its own authorization rules that can again be stored in the catalog.

1.5.9 Performance Utilities

Three key performance indicators (KPIs) of a DBMS are: response time; throughput rate; and space utilization. The response time denotes the time elapsed between issuing a database request (e.g., a query or update instruction) and the successful termination thereof. The throughput rate represents the transactions a DBMS can process per unit of time. Space utilization refers to the space utilized by the DBMS to store both the raw data and the metadata. A high-performing DBMS is characterized by quick response times, high throughput rates, and low space utilization.

DBMSs come with various types of utilities aimed at improving these three KPIs. Examples are utilities to distribute and optimize data storage, to tune indexes for faster query execution, to tune queries to improve application performance, or to optimize buffer management (buffering is instrumental to the exchange of data and updates between internal memory and disk storage). These utilities are typically managed by the DBA.

Retention Questions

- What are the advantages of database systems and database management?
- What is data independence and why is it needed?
- What are integrity rules? Illustrate with examples.
- What is the difference between structured, semi-structured, and unstructured data?
- Define the ACID properties in a transaction management context.