Database technology is not only improving the daily operations of organizations but also the quality of decisions that affect our lives. Databases contain a flood of data about many aspects of our lives: consumer preferences, telecommunications usage, credit history, television viewing habits, and so on.

Database technology helps to summarize this mass of data into useful information for making decision. Management uses information gleaned from databases to make long-range decisions such as investing in plants and equipment, locating stores, adding new items to inventory, and entering new businesses.

This chapter provides a starting point for your exploration of Database Technology. It surveys database characteristics, database management system features, system architectures, and human roles in managing and using databases. This chapter provides a broad picture of database technology.

3.1 DATABASE CHARACTERISTICS

Every day, businesses collect mountains of facts about persons, things, and events such as credit card numbers, bank balances, and purchase amounts. Databases contain these sorts of simple facts as well as unconventional information like photographs, fingerprints, product videos, and book abstracts. With the proliferation of the Internet and the means to capture data in computerized form, a vast amount of data is available at the click of a mouse button.
With so much data available, organizing these data for ease of retrieval and maintenance is paramount. Thus, managing databases has become a vital task in organizations.

Database a collection of persistent data that can be shared and interrelated.

Before learning about managing databases, we must first understand some important properties of databases:

Persistent means that data reside on stable storage such as a magnetic disk. For example, organizations need to retain data about customers, suppliers, and inventory on stable storage because these data are repetitively used. A variable in a computer program is not persistent because it resides in main memory and disappears after the program terminates. Persistency does not mean that data last forever. When data are no longer relevant (such as a supplier going out of business), they are removed or archived.

Persistency depends on relevance of intended usage. For example, the mileage you drive for work is important to maintain if you are self-employed. Likewise, the amount of your medical expenses is important if you can itemize your deductions.

Because storing and maintaining data is costly, only data likely to be relevant to decisions should be stored.

"Shared" means that a database can have multiple uses and users. A database provides a common memory for multiple functions in an organization. For example, a personnel database can support payroll calculations, performance evaluations, government reporting requirements, and so on. Many users can use a database at the same time. For example, many customers can simultaneously make airline reservations. Unless two users are trying to change the same part of the database at the same time, they can proceed without waiting on each other.

Interrelated means that data stored as separate units can be connected to provide a whole picture. For example, a customer database relates customer data (name, address, . . .) to order data (order number, order date . . .) to facilitate order processing. Databases contain both entities and relationships among entities. An **entity** is a cluster of data usually about one topic that can be accessed together. An entity can denote a person, place, thing, or event. For example, a personnel database contains entities such as employees, departments, and skills as well as relationships showing employee assignments to departments, skills possessed by employees, and salary history of employees. A typical business database may have hundreds of entities and relationships.

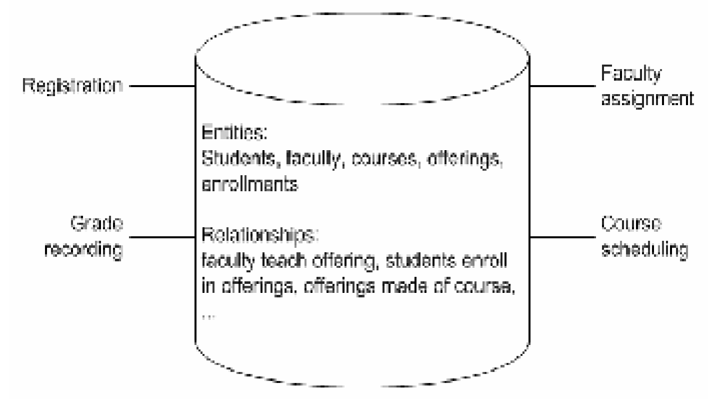

Registration
Faculty assignment
Entities:
Students, faculty, courses, offerings, enrollments
Grade recording
Relationships:
faculty teach offering, students enroll in offerings, offerings made of course, ...
Course scheduling

Figure 1 Depiction of a simplified university database

To depict these characteristics, let us consider a number of databases. A simplified university database contains data about students, faculty, courses, course offerings, and enrollments. The database supports procedures such as registering for classes, assigning faculty to course offerings, recording grades, and scheduling course offerings. Relationships in the university database support answers to questions such as:

- What offerings are available for a course in a given academic period?
- Who is the instructor for an offering of a course?
- What students are enrolled in an offering of a course?

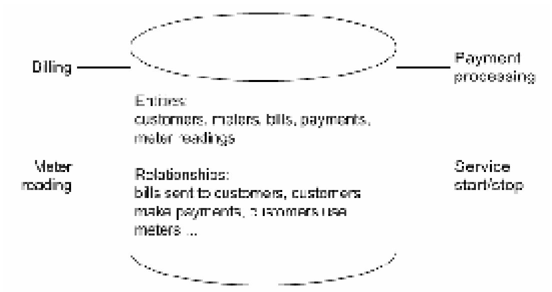

Billing
Payment processing
Entities:
customers, meters, bills, payments, meter readings
Meter reading
Relationships:
bills sent to customers, customers make payments, customers use meters ...
Service start/stop

Figure 2 Depiction of a simplified water utility database

Next, let us consider a water utility database as depicted in Figure 2. The primary function of a water utility database is billing customers for water usage. Periodically, a customer's water consumption is measured from a meter and a bill is prepared. Many aspects can influence the preparation of a bill such as a customer's payment history, meter characteristics, type of customer (low income, renter, homeowner, small business, large business, etc.), and billing cycle. Relationships in the water utilities database support answers to questions such as:

- What is the date of the last bill sent to a customer?
- How much water usage was recorded when a customer's meter was last read?
- When did a customer make his/her last payment?

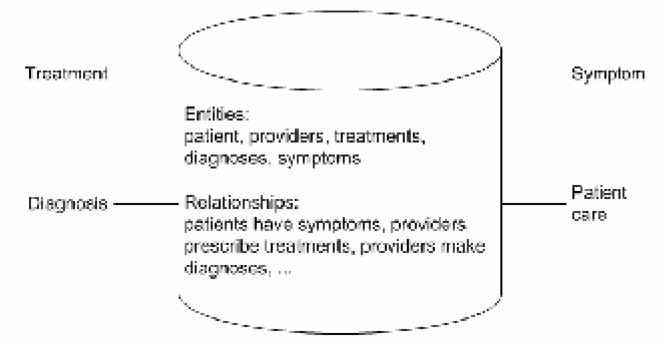

Treatment
Symptom
Entities:
patient, providers, treatments, diagnoses, symptoms
Diagnosis
Relationships:
patients have symptoms, providers prescribe treatments, providers make diagnoses, ...
Patient care

Figure 3 Depiction of a simplified hospital database

Finally, let us consider a hospital database as depicted in Figure 3. The hospital database supports treatment of patients by physicians. Physicians make diagnoses and prescribe treatments based on symptoms. Many different health providers read and contribute to a patient's record. Nurses are responsible for monitoring symptoms and providing medication. Food workers prepare meals according to a treatment plan. Physicians prescribe new treatments based on the results of previous treatments and patient symptoms. Relationships in the database support answers to questions such as:

- What are the most recent symptoms of a patient?
- Who prescribed a given treatment of a patient?
- What diagnosis did a doctor make for a patient?

These simplified databases lack many kinds of data found in real databases. For example, the simplified university database does not contain data about course prerequisites and classroom capacities and locations. Real versions of these databases would have many more entities and additional uses. Nevertheless, these simple databases have the essential characteristics of business databases: persistent data, multiple users and uses, and multiple entities connected by relationships.

3.2 FEATURES OF DBMS

A database management system (DBMS) is a collection of software that supports the creation, use, and maintenance of databases. Initially, DBMSs provided efficient storage and retrieval of data. Due to marketplace demands and product innovation, DBMSs have evolved to provide a broad range of features for data acquisition, storage, dissemination, maintenance, retrieval, and formatting. The evolution of these features has made DBMSs rather complex. It can take years of study and use to master a particular DBMS. Because DBMSs continue to evolve, you must continually update your knowledge.

Database Management System (DBMS) a collection of components that support data acquisition, dissemination, maintenance, retrieval, and formatting.

To provide insight about features that you will encounter in commercial DBMSs, Table 1 (appendix) summarizes a common set of features. The remainder of this section presents examples of these features. Some examples are drawn from Microsoft Access, a popular desktop DBMS.

Database Definition

To define a database, the entities and relationships must be specified. In most commercial DBMSs, tables store collections of entities. Table 2 (in appendix) has a heading row (first row) showing the column names and a body (other rows) showing the contents of the table. Relationships indicate connections among tables. For example, the relationship connecting the student table to the enrollment table shows the course offerings taken by each student.

Table a named, two-dimensional arrangement of data, consists of heading part and a body part.

Most DBMSs provide several tools to define databases. The Structured Query Language (SQL) is an industry standard language supported by most DBMSs. SQL can be used to define tables, relationships among tables, integrity constraints (rules that define allowable data), and authorization rights (rules that restrict access to data).

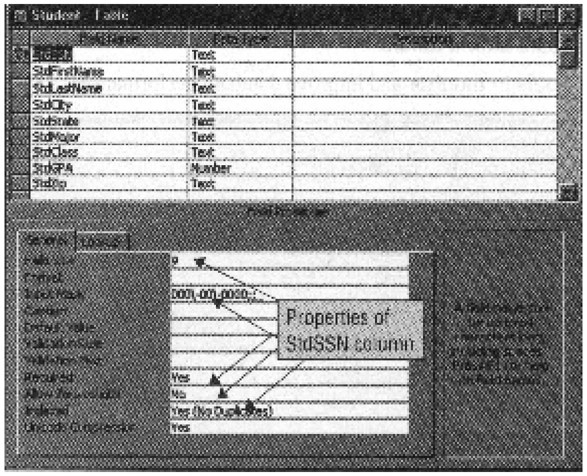


Figure 4 Table definition window in Microsoft Access

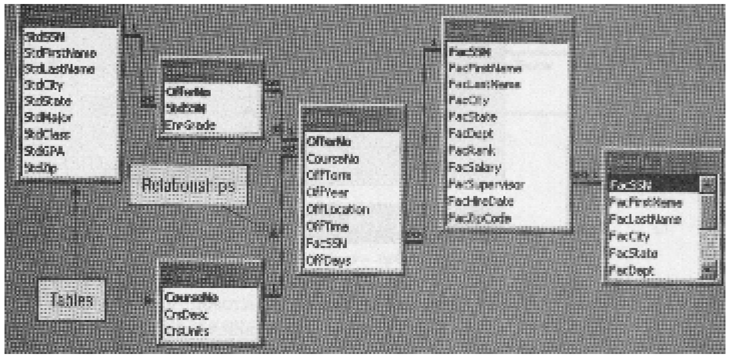


Figure 5 Relationship definition window in Microsoft Access

In addition to SQL, many DBMSs provide graphical, window-oriented tools. Figures 4 and 5 depict graphical tools for defining tables and relationships. Using the Table Definition window in Figure 4, the user can define properties of columns such as the data type and field size. Using the Relationship Definition window in Figure 6, relationships among tables can be defined. After defining the structure, a database can be populated. The data in Table 2 (in appendix) should be added after the Table Definition window and Relationship Definition window are complete.

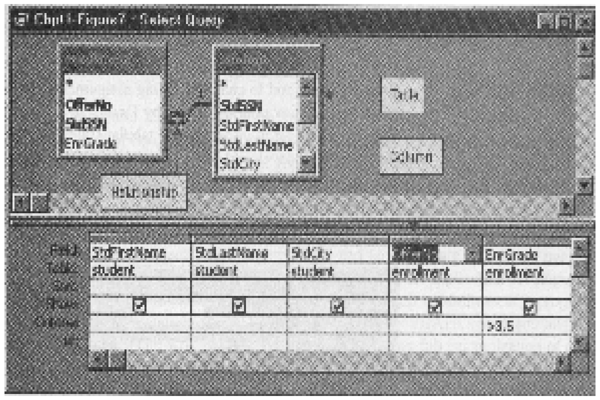


Figure 6 Query design window in Microsoft Access

SQL an industry standard database language that includes statements for database definition, database manipulation, and database control.

Nonprocedural Access

The most important feature of DBMSs is the ability to answer queries. A query is a request for data to answer a question. For example, the user may want to know customers having large balances or products with strong sales in a particular region. Nonprocedural access allows users with limited computing skills to submit queries. The user specifies what parts of a database to retrieve, not the details of how retrieval occurs. The "how" part involves coding complex procedures with loops.

Nonprocedural Database Language a language such as SQL that allows you to specify what part of a database to access rather than to code a complex procedure. Nonprocedural languages do not include looping statements.

Nonprocedural languages do not have looping statements (for, while, and so on) because only the "what" part is specified.

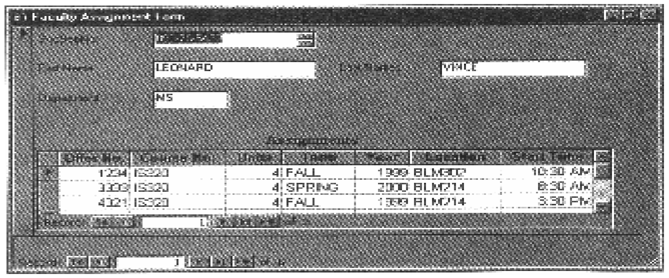
Nonprocedural access can reduce the number of lines of code by a factor of 100 as compared to procedural access. Because a large part of business software involves data access, nonprocedural access can provide a dramatic improvement in software productivity.

To appreciate the significance of nonprocedural access, consider an analogy to planning a vacation. You specify your destination, travel budget, length of stay, and departure date. These facts indicate the "what" of your trip. To specify the "how" of your trip, you need to indicate many more details such as the best route to your destination, the most desirable hotel, ground transportation, and so on. Your planning process is much easier if you have a professional to help with these additional details. Like a planning professional, a DBMS performs the detailed planning process to answer queries expressed in a nonprocedural language.

Most DBMSs provide more than one tool. For nonprocedural access, the SELECT statement of SQL, described provides a nonprocedural way to access a database. Most DBMSs also provide graphical tools to access databases. Figure 6 depicts a graphical tool available in Microsoft Access. To pose a query to the database, a user only has to indicate the required tables, relationships, and columns. Access is responsible for knowing how to retrieve the requested data. Table 3 (in appendix) shows the result of executing the query in Figure 6.

Application Development and Procedural Language Interface

Most DBMSs go well beyond simply accessing data. Graphical tools are provided for building complete applications using forms and reports. Data entry forms provide a convenient way to enter and edit data, while reports enhance the appearance of data that are displayed or printed. The form in Figure 7 can be used to add new course assignments for a professor and to change existing assignments. The report in Figure 8 uses indentation to show courses taught by faculty in various departments. The indentation style can be easier to view than the tabular style shown in Table 3.


Faculty Assignment Form

First Name: LEONARD Last Name: VINCE
Department: MS

Assignments

	Offer No.	Course No.	Units	Term	Year	Location	Start Time
▶	1234	IS320	4	FALL	1999	BLM302	10:30 AM
	3333	IS320	4	SPRING	2000	BLM214	8:30 AM
	4321	IS320	4	FALL	1999	BLM214	3:30 PM

Figure 7 Microsoft Access form for assigning courses to faculty

Many forms and reports can be developed with a graphical tool without detailed coding. For example, Figures 7 and 8 were developed without coding.

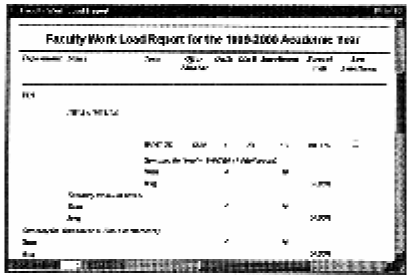

Faculty Work Load Report for the 1999-2000 Academic Year

Figure 8 Microsoft Access report faculty workload

Nonprocedural access makes form and report creation possible without extensive coding. As part of creating a form or report, the user indicates the data requirements using a nonprocedural language (SQL) or graphical tool. To complete a form or report definition, the user indicates formatting of data, user interaction, and other details.

In addition to application development tools, a procedural language interface adds the full capabilities of a computer programming language.

Procedural Language Interface a method to combine a nonprocedural language such as SQL with programming language such as COBOL or visual Basic.

Nonprocedural access and application development tools, though convenient and powerful, are sometimes not efficient enough or do not provide the level of control necessary for application development. When these tools are not adequate, DBMSs provide the full capabilities of a programming language. For example, Visual Basic for Applications (VBA) is a programming language that is integrated with Microsoft Access. VBA allows full customization of database access, form processing, and report generation. Most commercial DBMSs have a procedural language interface comparable to VBA. For example, Oracle has the language PL/SQL and Microsoft SQL Server has the language Transact-SQL.

Other Features

Transaction processing enables a DBMS to process large volumes of repetitive work. A transaction is a unit of work that should be processed reliably without interference from other users and without loss of data due to failures. Examples of transactions are withdrawing cash at an ATM, making an airline reservation, and registering for a course. A DBMS ensures that transactions are free of interference from other users, parts of a transaction are not lost due to a failure, and transactions do not make the database inconsistent. Transaction processing is largely a "behind the scenes" affair.

Transaction Processing reliable and efficient processing of large volumes of repetitive work. DBMSs ensure that simultaneous users do not interfere with each other and that failures do not cause lost work.

The user does not know the details about transaction processing other than the assurances about reliability.

Database tuning includes a number of monitors and utility programs to improve performance.

Some DBMSs can monitor how a database is used, the distribution of various parts of database, and the growth of the database. Utility programs can be provided to reorganize a database, select physical structures for better performance, and repair damaged parts of a database.

Transaction processing and database tuning are most prominent on DBMSs that support large databases with many simultaneous users. These DBMSs are known as enterprise DBMSs because the databases they support are often critical to the functioning of an organization.

Enterprise DBMSs usually run on powerful servers and have a high cost. In contrast, desktop DBMSs running on personal computers and small servers support limited transaction processing features but have a much lower cost. Desktop DBMSs support databases used by work teams and small businesses.

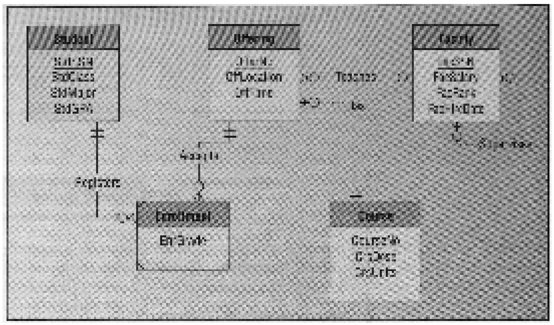


Figure 9 Entity relationship diagram (ERD) created with Visio Professional

In addition to features provided directly by vendors of DBMSs, third-party software is also available for many DBMSs. In most cases, third-party software extends the features available with the database software. For example, many third-party vendors provide advanced database design tools that extend the database definition and tuning capabilities provided by DBMSs. Figure 9 shows a database diagram (an entity relationship diagram) created with Visio Professional, a tool for database design. The ERD in Figure 9 can be converted into the tables supported by most commercial DBMSs. In some cases, third-party software competes directly with the database product. For example, third-party vendors provide application development tools that can be used in place of the ones provided with the database product.

3.3 DEVELOPMENT OF DATABASE TECHNOLOGY AND MARKET STRUCTURE

The previous section provided a quick tour of the features found in typical DBMSs. The features in today's products are a significant improvement over just a few years ago. Database management, like many other areas of computing, has undergone tremendous technological growth. To provide you a context to appreciate today's DBMSs, this section reviews past changes in technology and suggests future trends. After this review, the current market for database software is presented.

Evolution of Database Technology

Table 4 depicts a brief history of database technology through four generations' of systems. The first generation supported sequential and random searching, but the user was required to write a computer program to obtain access. For example, a program could be written to retrieve all customer records or to just find the customer record with a specified customer number. Because first-generation systems did not offer much support for relating data, they are usually regarded as file processing systems rather than DBMSs. File processing systems can manage only one entity rather than many entities and relationships managed by a DBMS.

The second-generation products were the first true DBMSs as they could manage multiple entity types and relationships. However, to obtain access to data, a computer program still had to be written. Second-generation systems are referred to as "navigational" because the programmer had to write code to navigate among a network of linked records. Some of the second-generation products adhered to a standard database definition and manipulation language developed by the Committee on Data Systems Languages (CODASYL), a standards organization.

The CODASYL standard had only limited market acceptance partly because IBM, the dominant computer company during this time, ignored the standard. IBM supported a different approach known as the hierarchical data model.

Rather than focusing on the second-generation standard, research labs at
IBM and academic institutions developed the foundations for a new generation of DBMSs. The most important development involved nonprocedural languages for database access.

Third-generation systems are known as relational DBMSs because of the foundation based on mathematical relations and associated operators. Optimization technology was developed so that access using nonprocedural languages would be efficient. Because nonprocedural access provided such an improvement over navigational access, third-generation systems supplanted the second generation. Since the technology was so different, most of the new systems were founded by start-up companies rather than by vendors of previous generation products. IBM was the major exception. It was IBM's weight that led to adoption of SQL as a widely accepted standard.

Fourth-generation DBMSs are extending the boundaries of database technology to unconventional data and the Internet. Fourth-generation systems can store and manipulate unconventional data types such as images, videos, maps, sounds, and animations. Because these systems view any kind of data as an object to manage, fourth-generation systems are sometimes called "object-oriented" or "object-relational." In addition to the emphasis on objects, the Internet is pushing DBMSs to develop new forms of distributed processing. Most DBMSs now feature convenient ways to publish static and dynamic data on the Internet. The market for fourth-generation systems is a battle between vendors of third-generation systems who are upgrading their products against a new group of systems developed by start-up companies. So far, the existing companies seem to have the upper hand.

Current Market for Database Software

According to Dataquest, a division of the Gartner Group, the sales of enterprise database software reached $7.1 billion in 1998, a 15 percent gain over 1997. Enterprise DBMSs use mainframe servers running IBM's MVS operating system and midrange servers running the Unix and Microsoft NT operating systems. Electronic commerce on the Internet has rejuvenated the market for enterprise database software. Before the advent of electronic commerce, enterprise DBMSs were becoming commodities and sales growth was stagnant. Dataquest projects sales of enterprise DBMSs to reach $10 billion by 2003.

According to Dataquest, five products dominate the market for enterprise database software as shown in Table 4. Although IBM holds the largest market share, Oracle dominates in the faster-growing Unix and NT markets. The overall market is very competitive with the major companies and smaller companies introducing many new features with each release.

In the market for desktop database software, Microsoft Access dominates at least in part because of the dominance of Microsoft Office. Desktop database software is primarily sold as part of office productivity software. With Microsoft Office holding about 90 percent of the office productivity market, Access holds a comparable share of the desktop database software market. Other significant products in the desktop database software market are Paradox, Approach, FoxPro, and FileMaker Pro.

3.4 ARCHITECTURES OF DATABASE MANAGEMENT SYSTEMS

To provide insight about the internal organization of DBMSs, this section describes two architectures or organizing frameworks. The first architecture describes an organization of database definitions to reduce the cost of software maintenance. The second architecture describes an organization of data and software to support remote access. These architectures promote a conceptual understanding rather than indicate how an actual DBMS is organized.

Data Independence and the Three Schema Architecture

In early DBMSs, there was a close connection between a database and computer programs that accessed the database. Essentially, the DBMS was considered part of a programming language. As a result, the database definition was part of the computer programs that accessed the database. In addition, the conceptual meaning of a database was not separate from its physical implementation on magnetic disk. The definitions about the structure of a database and its physical implementation were mixed inside computer programs.

Data Independence a database should have an identity separate from the applications (computer programs, forms, and reports) that use it. The separate identity allows the database definition to be changed without affecting related applications.

The close association between a database and related programs led to problems in software maintenance. Software maintenance encompassing requirement changes, corrections, and enhancements can consume a large fraction of computer budgets. In early DBMSs, most changes to the database definition caused changes to computer programs. In many cases, changes to computer programs involved detailed inspection of the code, a labor-intensive process. This code inspection work is similar to "year 2000 compliance" where date formats must be changed to four digits.

Performance tuning of a database was difficult because sometimes hundreds of computer programs had to be recompiled for every change. Because database definition changes are common, a large fraction of software maintenance resources was devoted to database changes. Some studies have estimated the percentage as high as 50 percent of software maintenance resources.

The concept of data independence emerged to alleviate problems with program maintenance. Data independence means that a database should have an identity separate from the applications (computer programs, forms, and reports) that use it. The separate identity allows the database definition to be changed without affecting related applications. For example, if a new column is added to a table, applications not using the new column should not be affected. Likewise if a new table is added, only applications that need the new table should be affected. This separation should be even more pronounced if a change only affects physical implementation of a database. Database specialists should be free to experiment with performance tuning without worrying about making computer program changes.

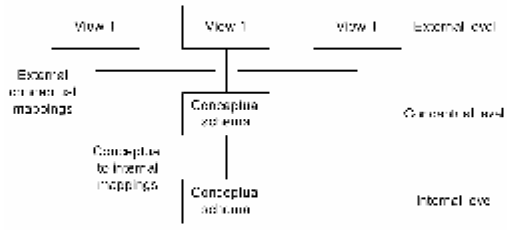


Figure 10 Three schema architecture

In the mid 1970s, the concept of data independence led to the proposal of the Three Schema Architecture depicted in Figure 10. The word schema as applied to databases means database description. The Three Schema Architecture includes three levels of database description. The external level is the user level. Each group of users can have a separate external view (or view for short) of a database tailored to the group's specific needs.

Three Schema Architecture anarchitecture for compartmentalizing database descriptions. The Three Schema architecture was proposed as a way to achieve data independence.

In contrast, the conceptual and internal schemas represent the entire database. The conceptual schema defines the entities and relationships. For a business database, the conceptual schema can be quite large, perhaps hundreds of entity types and relationships. Like the conceptual schema, the internal schema represents the entire database. However, the internal schema represents the storage view of the database whereas the conceptual schema represents the logical meaning of the database. The internal schema defines files, collections of data on a storage device such as a hard disk. A file can store one or more entities described in the conceptual schema.

To make the three schema levels clearer, Table 5 shows differences among database definition at the three schema levels using examples from the features described in Section 3.2. Even in a simplified university database, the differences among the schema levels are clear. With a more complex database, the differences would be even more pronounced with many more views, a much larger conceptual schema, and a more complex internal schema.

The schema mappings describe how a schema at a higher level is derived from a schema at a lower level. For example, the external views in Table 5 are derived from the tables in the conceptual schema. The mapping provides the knowledge to convert a request using an external view (for example, HighGPAView) into a request using the tables in the conceptual schema. The mapping between conceptual and internal levels shows how entities are stored in files.

DBMSs, using schemas and mappings, ensure data independence. Typically, applications access a database using a view. The DBMS converts an application's request into a request using the conceptual schema rather than the view. The DBMS then transforms the conceptual schema request into a request using the internal schema. Most changes to the conceptual or internal schema do not affect applications because applications do not use the lower schema levels. The DBMS, not the user, is responsible for using the mappings to make the transformations.

The Three Schema Architecture is an official standard of the American National Standards Institute (ANSI). However, the specific details of the standard were never widely adopted. Rather, the standard serves as a guideline about how data independence can be achieved. The spirit of the Three Schema Architecture is widely implemented in third- and fourth-generation DBMSs.

Distributed Processing and the Client-Server Architecture

With the growing importance of network computing and the Internet, distributed processing is becoming a crucial function of DBMSs. Distributed processing allows geographically dispersed computers to cooperate when providing data access. A

large part of electronic commerce on the Internet involves accessing and updating remote databases. Many databases in retail, banking, and security trading are now available through the Internet. DBMSs use available network capacity and local processing capabilities to provide efficient remote database access.

Many DBMSs support distributed processing using client-server architecture. A client is a program that submits requests to a server. A server processes requests on behalf of a client. For example, a client may request a server to retrieve product data. The server locates the data and sends them back to the client. The client may perform additional processing on the data before displaying the results to the user. As another example, a client submits a completed order to a server. The server validates the order, updates a database, and sends an acknowledgement to the client. The client informs the user that the order has been processed.

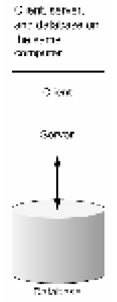


Figure 10 Client/server database on same computer

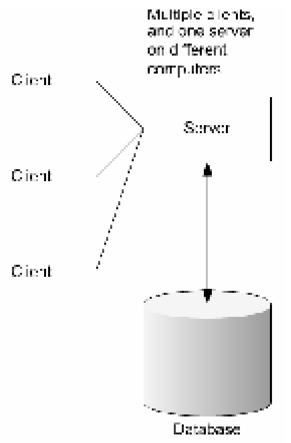


Figure 11 Multiple clients and one server on different computers

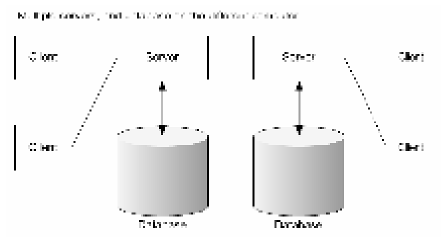


Figure 12 Multiple servers and databases on different computers

To improve performance and availability of data, the client-server architecture supports many ways to distribute software and data in a computer network. The simplest scheme is just to place both software and data on the same computer, Figure 10. To take advantage of a network, both software and data can be distributed. In Figure 11, the server software and the database are located on a remote computer. In Figure 12, the server software and the database are located on multiple remote computers.

Client-Server Architecture an arrangement of components (clients and servers) and data among computers connected by a network. The client-server architecture supports efficient processing of messages (requests for service) between clients and servers.

The DBMS has a number of responsibilities in client-server architecture. The DBMS provides software that can execute on both the client and the server. The client software is typically responsible for accepting user input, displaying results, and performing some processing of data. The server software validates client requests, locates remote databases, updates remote databases (if needed), and sends the data in a format that the client understands.

Client-server architectures provide a flexible way for DBMSs to interact with computer networks. The distribution of work among clients and servers and the possible choices to locate data and software are much more complex than described here.

3.5 ORGANIZATIONAL IMPACTS OF DATABASE TECHNOLOGY

This section completes your introduction to database technology by discussing how databases affect organizations. The first section describes how you might interact with a database in an organization. The second section describes information resource management, an effort to control the data produced and used by an organization. Special attention is given to management roles that you may play as part of an effort to control information resources.

Interacting with Databases

Because databases are pervasive, there are a variety of ways in which you may interact with databases. The classification in Figure 13 distinguishes between functional users who interact with databases as part of their work and information systems professionals who participate in designing and implementing databases. Each box in the hierarchy represents a role that you may play. You may simultaneously play more than one role. For example, a functional user in a job such as financial analysis may play all three roles in different databases. In some organizations, the distinction between functional users and information systems professionals is blurred. In these organizations, functional users may participate in designing and using databases.

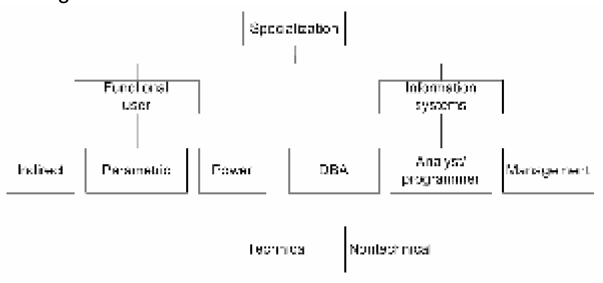


Figure 13 Classification of roles

Functional users can play a passive or an active role when interacting with databases. Indirect usage of a database is a passive role. An indirect user is given a report or some data extracted from a database. A parametric user is more active than an indirect user. A parametric user requests existing forms or reports using parameters, input values that change from usage to usage. For example, a parameter may indicate a date range, sales territory, or department name. The power user is the most active. Because decision-making needs can be difficult to predict, ad-hoc or unplanned usage of a database is important. A power user is skilled enough to build a form or report when needed.

Database Administrator a support position that specializes in managing individual databases and DBMSs.

Power users should have a good understanding of nonprocedural access, a skill described in the first part of this book. Information systems professionals interact with databases as part of developing an information system. Analyst/programmers are responsible for collecting requirements, designing applications, and implementing information systems. They create and use external views to develop forms, reports, and other parts of an information system. Management has an oversight role in development of databases and information systems.

Database administrators assist both information systems professionals and functional users.

Database administrators have a variety of both technical and non-technical responsibilities Table 6. Technical skills are more detail-oriented; non-technical responsibilities are more people-oriented. The primary technical responsibility is database design. On the non-technical side, the database administrator's time is split among a number of activities. Database administrators also can have responsibilities in planning databases and evaluating DBMSs.

APPENDIX

Table 1 Summary of common features of DBMSs

Features	Description
Database definition	Language and graphical tools to define entities, relationship, integrity constraints, and authorization rights.
Nonprocedural access	Language and graphical tools to access data without complicated coding.
Application development	Graphical tools to develop menus, data entry forms, and reports.
Procedural language interface	Language that combines nonprocedural access with full capabilities of a programming language.
Transaction processing	Control mechanisms to prevent interference from simultaneous users and recover lost data after a failure.
Database tuning	Tools to monitor and improve database performance.

Table 2 Display of student table in Microsoft Access

StdFirstName	StdLastName	StdCity	StdState	StdZip	StdMajor	STdClass	StdGPA
HOMER	WELLS	SEATTLE	WA	98121-1111	IS	FR	3.00
BOB	NORBERT	BOTHELL	WA	98011-2121	FIN	JR	2.70
CANDY	KENDALL	TACOMA	WA	99042-3321	ACCT	JR	3.50
WALLY	KENDALL	SEATTLE	WA	98123-1141	IS	SR	2.80
JOE	ESTRADA	SEATTLE	WA	98121-2333	FIN	SR	3.20
MARIAH	DODGE	SEATTLE	WA	98114-0021	IS	JR	3.60
TESS	DODGE	REDMOND	WA	98116-2344	ACCT	SO	3.30

Table 3 Result of executing query in Figure 6

StdFirstName	StdLastName	StdCity	OfferNo	EnrGrade
MARIAH	DODGE	SEATTLE	1234	3.8
BOB	NORBERT	BOTHELL	5679	3.7
ROBERTO	MORALES	SEATTLE	5679	3.8
MARIAH	DODGE	SEATTLE	6666	3.6
LUKE	BRAZZI	SEATTLE	7777	3.7
WILLIAM	PILGRIM	BOTHELL	9876	4.0

Table 4 Brief Evolution of Database Technology

Era	Generation	Orientation	Major Features
1960s	1st	File	File structures and proprietary program interfaces
1970s	2nd	Network navigation	Networks and hierarchies of related records, standard program interfaces
1980s	3rd	Relational	Nonprocedural languages, optimization, transaction processing
1990s	4th	Object	Multimedia, active, distributed processing, more powerful operators

Table 5 University database example depicting differences among schema levels

Schema Level	Description
External	HighGPAView: data required for the query in Figure 6.
	FacultyAssignmentFormView: data required for the form in Figure 7.
	FacultyWorkLoadReportView: data required for the report in Figure 8
Conceptual	Student, Enrollment, Course, Faculty, and Enrollment tables and relationship Figure 5.
Internal	Files needed to store the tables; extra files (indexed property in Figure 4) to improve performance

Table 6 Responsibilities of the database administrator

Technical	Nontechnical
Designing conceptual schemas Designing internal schemas Monitoring database performance Selecting and evaluating database software Designing client-server databases Troubleshooting database problems	Setting database standards Devising training materials Promoting benefits of databases Consulting with users