Tikrit University
College of Computer Science and Mathematics
Department of Computer Science

Database Basics
Lecture 2

Assistant.Lec. Mustafa Latif

Second Class

Lecture 2
Database System Concepts and Architecture

Outline

- Data Models, Schemas, and Instances
- Three-Schema Architecture and Data Independence
- Database Languages and Interfaces
- The Database System Environment
- Centralized and Client/Server Architectures for DBMSs
- Classification of Database Management Systems

2.1 Data Models, Schemas, and Instances

One fundamental characteristic of the database approach is that it provides some level of data abstraction.

Data abstraction generally refers to the suppression of details of data organization and storage, and the highlighting of the essential features for an improved understanding of data. One of the main characteristics of the database approach is to support data abstraction so that different users can perceive data at their preferred level of detail.

A **data model**—a collection of concepts that can be used to describe the structure of a database—provides the necessary means to achieve this abstraction. By structure of a database we mean the data types, relationships, and constraints that apply to the data. Most data models also include a set of **basic operations** for specifying retrievals and updates on the database.

Database System Concepts and Architecture

- Basic client/server DBMS architecture
  - **Client module**
  - **Server module**

Data Models, Schemas, and Instances

- **Data abstraction**
  - Suppression of details of data organization and storage
  - Highlighting of the essential features for an improved understanding of data

Data Models, Schemas, and Instances (cont'd.)

- **Data model**
  - Collection of concepts that describe the structure of a database
  - Provides means to achieve data abstraction
  - **Basic operations**
    - Specify retrievals and updates on the database
  - **Dynamic aspect** or **behavior** of a database application
    - Allows the database designer to specify a set of valid operations allowed on database objects

2.1.1 Categories of Data Models

Many data models have been proposed, which we can categorize according to the types of concepts they use to describe the database structure.

High-level or conceptual data models provide concepts that are close to the way many users perceive data.

low-level or physical data models provide concepts that describe the details of how data is stored on the computer storage media, typically magnetic disks. Concepts provided by physical data models are generally meant for computer specialists, not for end users.

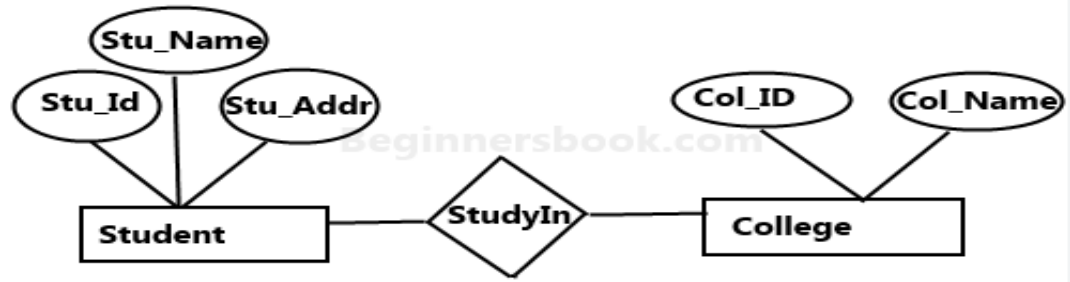
Between these two extremes is a class of **representational (or implementation) data models**, which provide concepts that may be easily understood by end users but that are not too far removed from the way data is organized in computer storage. Representational data models hide many details of data storage on disk but can be implemented on a computer system directly.

Conceptual data models use concepts such as entities, attributes, and relationships.

An **entity** represents a real-world object or concept, such as an employee or a project from the mini-world that is described in the database.

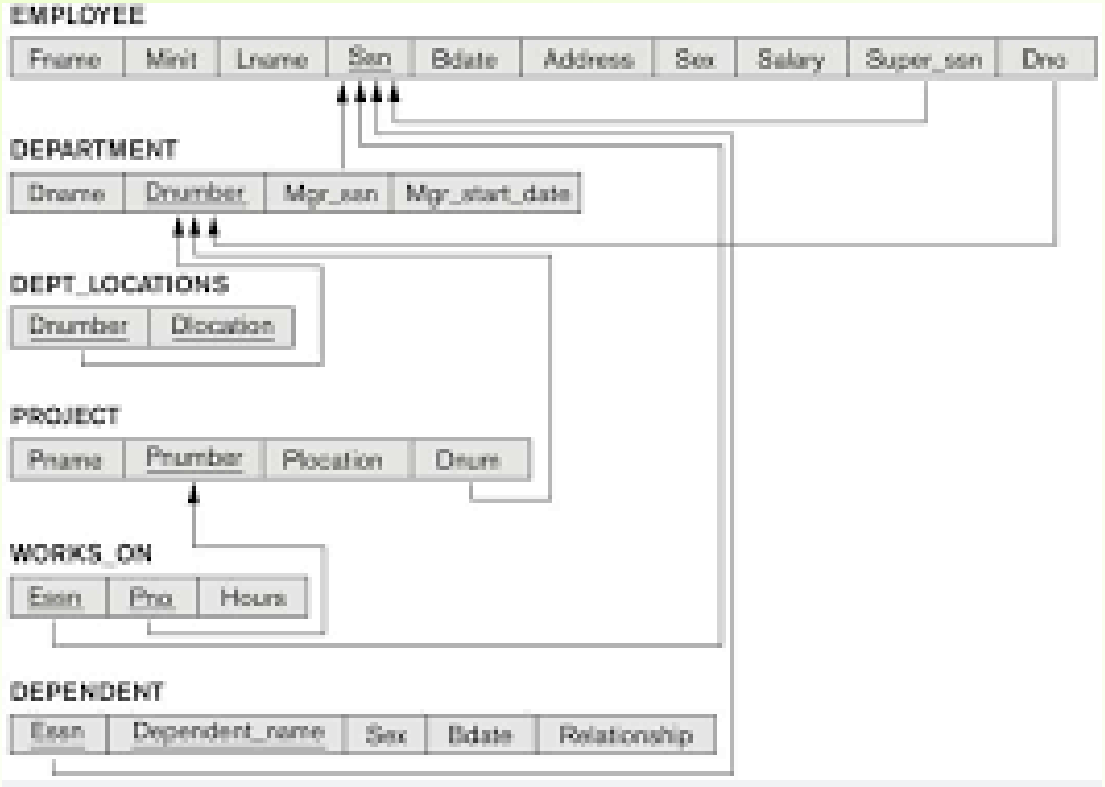
An **attribute** represents some property of interest that further describes an entity, such as the employee's name or salary.

A **relationship** among two or more entities represents an association among the entities, for example, a works-on relationship between an employee and a project.



Sample E-R Diagram

Representational or implementation data models are the models used most frequently in traditional commercial DBMSs. These include the widely used **relational data model**, as well as the so-called legacy data models—the **network** and **hierarchical models**—that have been widely used in the past.



Physical data models describe how data is stored as files in the computer by representing information such as record formats, orderings, and **access paths**. An access path is a search structure that makes the search for particular database records efficient, such as indexing or hashing. An **index** is an example of an access path that allows direct access to data using an index term or a keyword. It is similar to the index at the end of this text, except that it may be organized in a linear, hierarchical (tree-structured), or some other fashion.

Categories of Data Models

- **High-level** or **conceptual data models**
  - Close to the way many users perceive data
- **Low-level** or **physical data models**
  - Describe the details of how data is stored on computer storage media
- **Representational data models**
  - Easily understood by end users
  - Also similar to how data organized in computer storage

Categories of Data Models (cont'd.)

- **Entity**
  - Represents a real-world object or concept
- **Attribute**
  - Represents some property of interest
  - Further describes an entity
- **Relationship** among two or more entities
  - Represents an association among the entities
  - **Entity-Relationship model**

Categories of Data Models (cont'd.)

- **Relational data model**
  - Used most frequently in traditional commercial DBMSs
- **Object data model**
  - New family of higher-level implementation data models
  - Closer to conceptual data models

Categories of Data Models (cont'd.)

- **Physical data models**
  - Describe how data is stored as files in the computer
  - **Access path**
    - Structure that makes the search for particular database records efficient
  - **Index**
    - Example of an access path
    - Allows direct access to data using an index term or a keyword

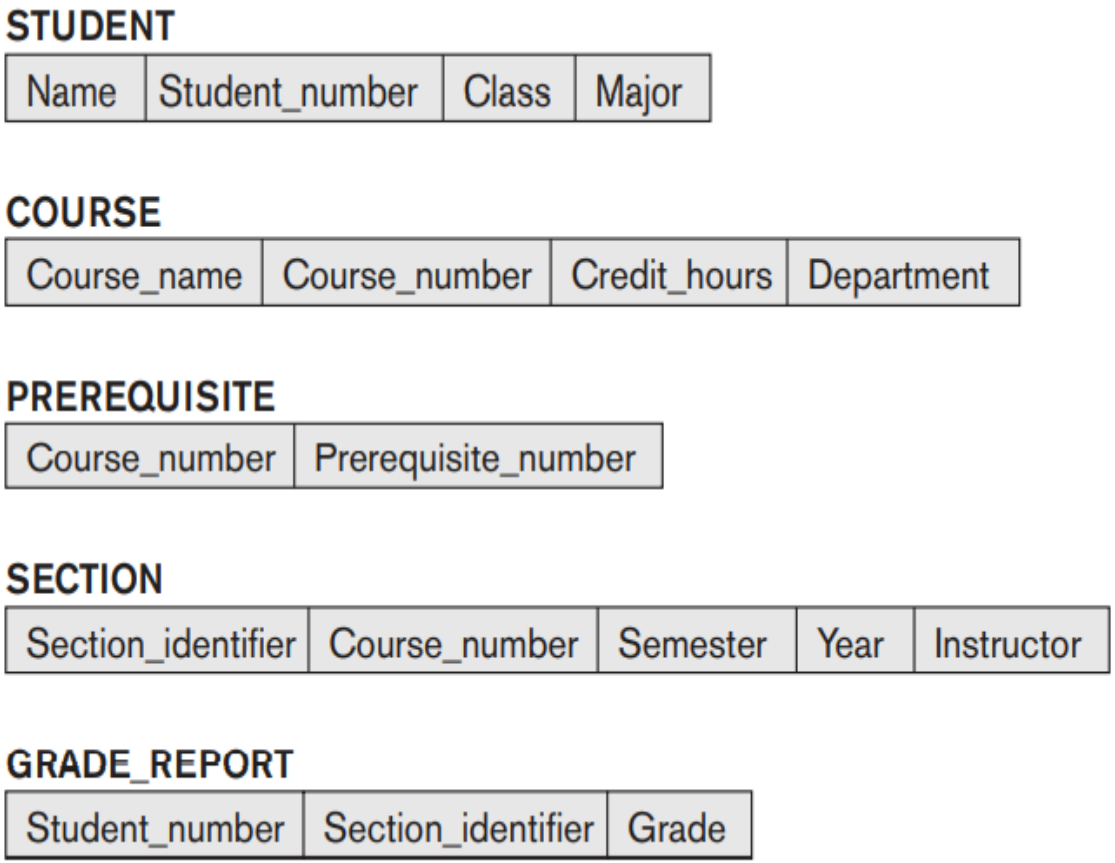
2.1.2 Schemas, Instances, and Database State

In a data model, it is important to distinguish between the description of the database and the database itself.

The description of a database is called the **database schema**, which is specified during database design and is not expected to change frequently. Most data models have certain conventions for displaying schemas as diagrams.

A displayed schema is called a **schema diagram**. Figure 2.1 shows a schema diagram for the database shown in Figure 2.1, the diagram displays the structure of each record type but not the actual instances of records. We call each object in the schema—such as **STUDENT** or **COURSE**—a **schema construct**.

Figure 2.1
Schema diagram for the database in Figure 1.2.



The actual data in a database may change quite frequently. For example, the database shown in Figure 2.1 changes every time we add a new student or enter a new grade.

The data in the database at a particular moment in time is called a **database state** or **snapshot**. It is also called the current set of **occurrences** or **instances** in the database.

In a given database state, each schema construct has its own current set of instances; for example, the **STUDENT** construct will contain the set of individual student entities (records) as its instances. Many database states can be constructed to correspond to a particular database schema. Every time we insert or delete a record or change the value of a data item in a record, we change one state of the database into another state.

2.2 Three-Schema Architecture and Data Independence

2.2.1 **The Three-Schema Architecture** The goal of the three-schema architecture, illustrated in Figure 2.2, is to separate the user applications from the physical database. In this architecture, schemas can be defined at the following three levels:

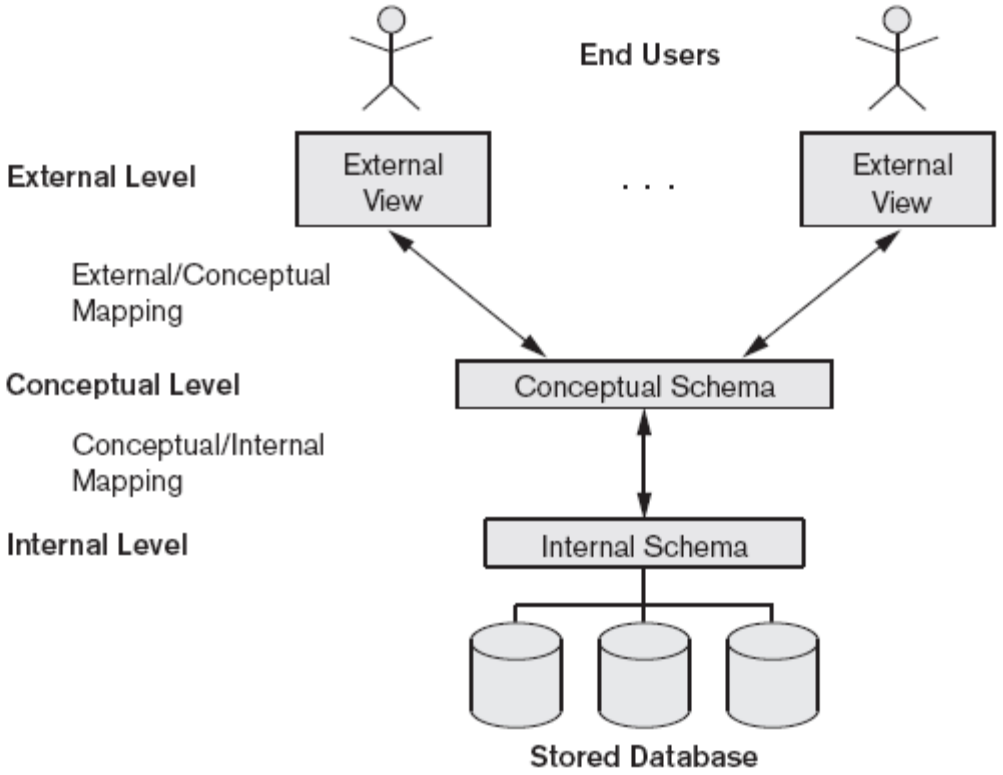
1. The **internal level** has an **internal schema**, which describes the physical storage structure of the database. The internal schema uses a physical data model and describes the complete details of data storage and access paths for the database.

2. The **conceptual level** has a conceptual schema, which describes the structure of the whole database for a community of users. The conceptual schema hides the details of physical storage structures and concentrates on describing entities, data types, relationships, user operations, and constraints. Usually, a representational data model is used to describe the conceptual schema when a database system is implemented. This implementation conceptual schema is often based on a conceptual schema design in a high-level data model.

3. The **external** or **view level** includes a number of external schemas or user views. Each external schema describes the part of the database that a particular user group is interested in and hides the rest of the database from that user group. As in the previous level, each external schema is typically implemented using a representational data model, possibly based on an external schema design in a high-level conceptual data model.

Three-Schema Architecture and Data Independence (cont'd.)

Figure 2.2
The three-schema architecture.



Schemas, Instances, and Database State

- **Database schema**
  - Description of a database
- **Schema diagram**
  - Displays selected aspects of schema
- **Schema construct**
  - Each object in the schema
- **Database state** or **snapshot**
  - Data in database at a particular moment in time

Schemas, Instances, and Database State (cont'd.)

- **Define** a new database
  - Specify database schema to the DBMS
- **Initial state**
  - **Populated** or **loaded** with the initial data
- **Valid state**
  - Satisfies the structure and constraints specified in the schema

Schemas, Instances, and Database State (cont'd.)

- **Schema evolution**
  - Changes applied to schema as application requirements change

Three-Schema Architecture and Data Independence

- **Internal level**
  - Describes physical storage structure of the database
- **Conceptual level**
  - Describes structure of the whole database for a community of users
- **External** or **view level**
  - Describes part of the database that a particular user group is interested in

2.2.2 Data Independence

The three-schema architecture can be used to further explain the concept of data independence, which can be defined as the capacity to change the schema at one level of a database system without having to change the schema at the next higher level. We can define two types of data independence: 1. Logical data independence is the capacity to change the conceptual schema without having to change external schemas or application programs. We may change the conceptual schema to expand the database (by adding a record type or data item), to change constraints, or to reduce the database (by removing a record type or data item). In the last case, external schemas that refer only to the remaining data should not be affected. For example, the external schema of Figure 1.5(a) should not be affected by changing the GRADE_REPORT file (or record type) shown in Figure 1.2 into the one shown in Figure 1.6(a). Only the view definition and the mappings need to be changed in a DBMS that supports logical data independence. After the conceptual schema undergoes a logical reorganization, application pro grams that reference the external schema constructs must work as before. Changes to constraints can be applied to the conceptual schema without affecting the external schemas or application programs.

Data Independence

- Capacity to change the schema at one level of a database system
  - Without having to change the schema at the next higher level
- Types:
  - **Logical**
  - **Physical**

DBMS Languages

- **Data definition language (DDL)**
  - Defines both schemas
- **Storage definition language (SDL)**
  - Specifies the internal schema
- **View definition language (VDL)**
  - Specifies user views/mappings to conceptual schema
- **Data manipulation language (DML)**
  - Allows retrieval, insertion, deletion, modification

DBMS Languages (cont'd.)

- **High-level** or **nonprocedural** DML
  - Can be used on its own to specify complex database operations concisely
  - **Set-at-a-time** or **set-oriented**
- **Low-level** or **procedural** DML
  - Must be embedded in a general-purpose programming language
  - **Record-at-a-time**

DBMS Interfaces

- Menu-based interfaces for Web clients or browsing
- Forms-based interfaces
- Graphical user interfaces
- Natural language interfaces
- Speech input and output
- Interfaces for parametric users
- Interfaces for the DBA

2.4 The Database System Environment

2.4.1 **DBMS Component Modules** Figure 2.3 illustrates, in a simplified form, the typical DBMS components. The figure is divided into two parts. The top part of the figure refers to the various users of the database environment and their interfaces. The lower part shows the internal modules of the DBMS responsible for storage of data and processing of transactions. The database and the DBMS catalog are usually stored on disk. Access to the disk is controlled primarily by the operating system (OS), which schedules disk read/write. Many DBMSs have their own buffer management module to schedule disk read/write, because management of buffer storage has a considerable effect on performance. Reducing disk read/write improves performance considerably. A higher-level stored data manager module of the DBMS controls access to DBMS information that is stored on disk, whether it is part of the database or the catalog. Let us consider the top part of Figure 2.3 first. It shows interfaces for the DBA staff, casual users who work with interactive interfaces to formulate queries, application programmers who create programs using some host programming languages, and parametric users who do data entry work by supplying parameters to predefined transactions. The DBA staff works on defining the database and tuning it by mak ing changes to its definition using the DDL and other privileged commands.

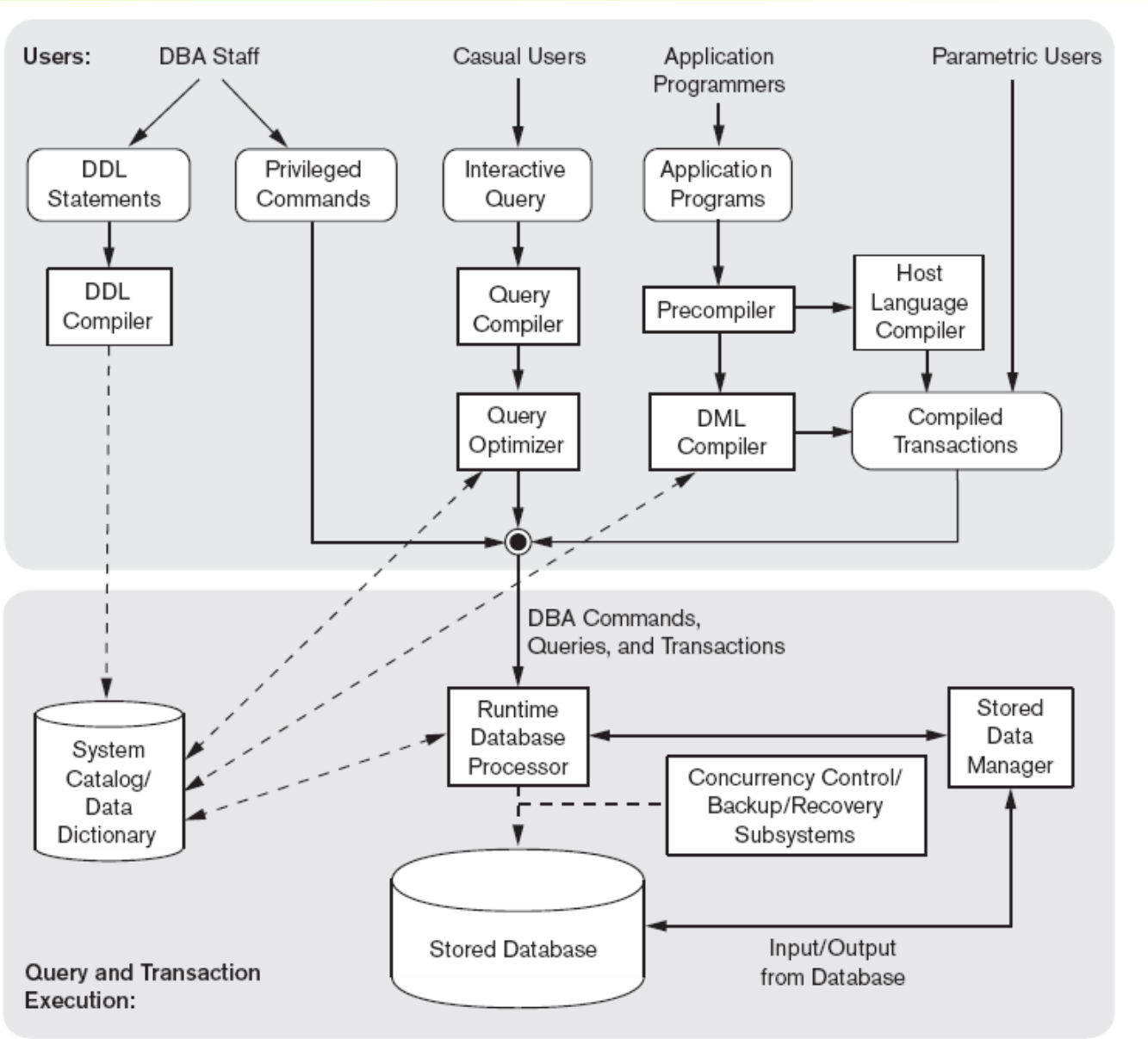


Figure 2.3
Component modules of a DBMS and their interactions.

The DDL compiler processes schema definitions, specified in the DDL, and stores descriptions of the schemas (meta-data) in the DBMS catalog. The catalog includes information such as the names and sizes of files, names and data types of data items, storage details of each file, mapping information among schemas, and constraints. In addition, the catalog stores many other types of information that are needed by the DBMS modules, which can then look up the catalog information as needed. Casual users and persons with occasional need for information from the database interact using the interactive query interface in Figure 2.3. We have not explicitly shown any menu-based or form-based or mobile interactions that are typically used to generate the interactive query automatically or to access canned transactions. These queries are parsed and validated for correctness of the query syntax, the names of files and data elements, and so on by a query compiler that compiles them into an internal form. This internal query is subjected to query optimization (discussed in Chapters 18 and 19). Among other things, the query optimizer is concerned with the rearrangement and possible reordering of operations, elimination of redundancies,

Among other things, the query optimizer is concerned with the rearrangement and possible reordering of operations, elimination of redundancies, and use of efficient search algorithms during execution. It consults the system catalog for statistical and other physical information about the stored data and generates executable code that performs the necessary operations for the query and makes calls on the runtime processor. Application programmers write programs in host languages such as Java, C, or C++ that are submitted to a precompiler. The precompiler extracts DML commands from an application program written in a host programming language. These commands are sent to the DML compiler for compilation into object code for database access. The rest of the program is sent to the host language compiler. The object codes for the DML commands and the rest of the program are linked, forming a canned transaction whose executable code includes calls to the runtime database processor. It is also becoming increasingly common to use scripting languages such as PHP and Python to write database programs. Canned transactions are executed repeatedly by parametric users via PCs or mobile apps; these users simply supply the parameters to the transactions. Each execution is considered to be a separate transaction. An example is a bank payment transaction where the account number, payee, and amount may be supplied as parameters.

In the lower part of Figure 2.3, the runtime database processor executes (1) the privileged commands, (2) the executable query plans, and (3) the canned transactions with runtime parameters. It works with the system catalog and may update it with statistics. It also works with the stored data manager, which in turn uses basic operating system services for carrying out low-level input/output (read/write) operations between the disk and main memory. The runtime database processor handles other aspects of data transfer, such as management of buffers in the main memory. Some DBMSs have their own buffer management module whereas others depend on the OS for buffer management. We have shown concurrency control and backup and recovery systems separately as a module in this figure. They are integrated into the working of the runtime database processor for purposes of transaction management.

2.4.2 **Database System Utilities** In addition to possessing the software modules just described, most DBMSs have database utilities that help the DBA manage the database system. Common utilities have the following types of functions:

- **Loading**. A loading utility is used to load existing data files—such as text files or sequential files—into the database. Usually, the current (source) format of the data file and the desired (target) database file structure are specified to the utility, which then automatically reformats the data and stores it in the database. With the proliferation of DBMSs, transferring data from one DBMS to another is becoming common in many organizations. Some vendors offer conversion tools that generate the appropriate loading programs, given the existing source and target database storage descriptions (internal schemas).

■ **Backup**. A backup utility creates a backup copy of the database, usually by dumping the entire database onto tape or other mass storage medium. The backup copy can be used to restore the database in case of catastrophic disk failure. Incremental backups are also often used, where only changes since the previous backup are recorded. Incremental backup is more complex, but saves storage space.

■ **Database storage reorganization**. This utility can be used to reorganize a set of database files into different file organizations and create new access paths to improve performance.

■ Performance monitoring. Such a utility monitors database usage and provides statistics to the DBA. The DBA uses the statistics in making decisions such as whether or not to reorganize files or whether to add or drop indexes to improve performance.

The Database System Environment

- DBMS component modules
  - **Buffer management**
  - **Stored data manager**
  - **DDL compiler**
  - **Interactive query** interface
    - **Query compiler**
    - **Query optimizer**
  - **Precompiler**

The Database System Environment (cont'd.)

- DBMS component modules
  - **Runtime database processor**
  - **System catalog**
  - **Concurrency control system**
  - **Backup and recovery system**

Database System Utilities

- **Loading**
  - Load existing data files
- **Backup**
  - Creates a backup copy of the database

Database System Utilities (cont'd.)

- **Database storage reorganization**
  - Reorganize a set of database files into different file organizations
- **Performance monitoring**
  - Monitors database usage and provides statistics to the DBA

Tools, Application Environments, and Communications Facilities

- CASE Tools
- **Data dictionary** (**data repository**) **system**
  - Stores design decisions, usage standards, application program descriptions, and user information
- **Application development environments**
- **Communications software**

2.5 Centralized and Client/Server Architectures for DBMSs

2.5.1 Centralized DBMSs Architecture

Architectures for DBMSs have followed trends similar to those for general computer system architectures. Older architectures used mainframe computers to provide the main processing for all system functions, including user application programs and user interface programs, as well as all the DBMS functionality. The reason was that in older systems, most users accessed the DBMS via computer terminals that did not have processing power and only provided display capabilities. Therefore, all processing was performed remotely on the computer system housing the DBMS, and only display information and controls were sent from the computer to the display terminals, which were connected to the central computer via various types of communications networks.

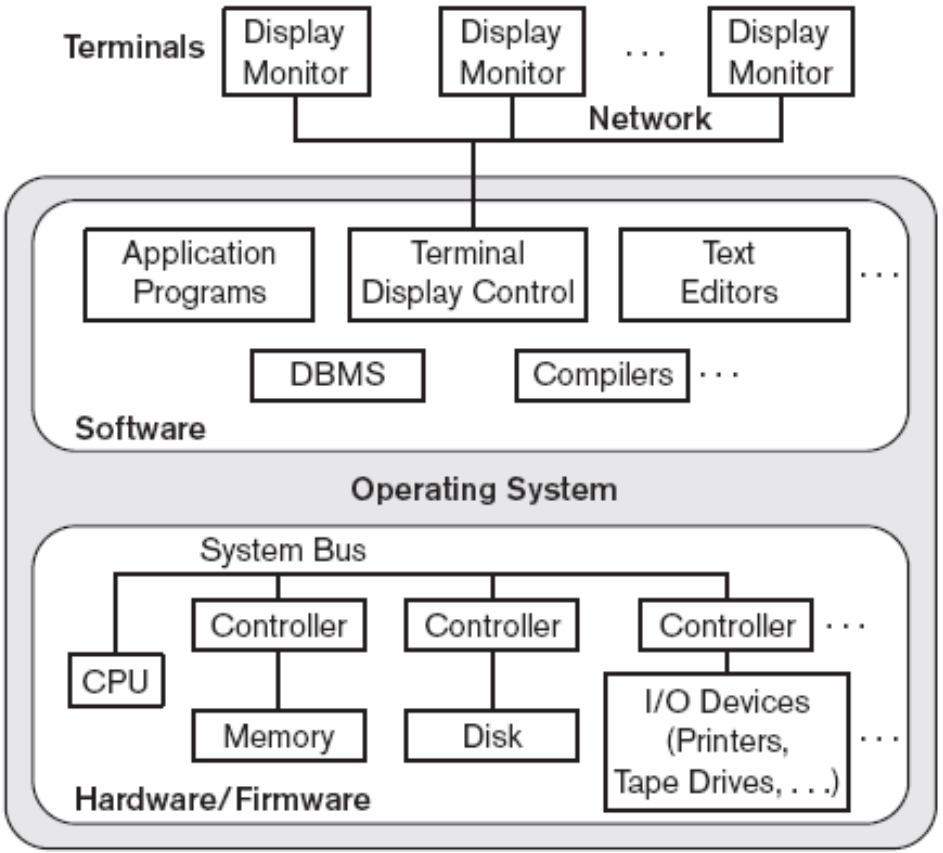


Figure 2.4
A physical centralized architecture.

2.5.2 **Basic Client/Server Architectures**

First, we discuss client/server architecture in general; then we discuss how it is applied to DBMSs. The client/server architecture was developed to deal with computing environments in which a large number of PCs, workstations, file servers, printers, database servers, Web servers, e-mail servers, and other software and equipment are connected via a network. The idea is to define specialized servers with specific functionalities. For example, it is possible to connect a number of PCs or small workstations as clients to a file server that maintains the files of the client machines. Another machine can be designated as a printer server by being connected to various printers; all print requests by the clients are forwarded to this machine. Web servers or e-mail servers also fall into the specialized server category.

The resources provided by specialized servers can be accessed by many client machines. The client machines provide the user with the appropriate interfaces to utilize these servers, as well as with local processing power to run local applications. This concept can be carried over to other software packages, with specialized programs—such as a CAD (computer-aided design) package—being stored on specific server machines and being made accessible to multiple clients.

that client and server software usually run on separate machines. Two main types of basic DBMS architectures were created on this underlying client/server framework: two-tier and three-tier. We discuss them next.

2.5.3 **Two-Tier Client/Server Architectures**

for DBMSs In relational database management systems (RDBMSs), many of which started as centralized systems, the system components that were first moved to the client side were the user interface and application programs. Because SQL (see Chapters 6 and 7) provided a standard language for RDBMSs, this created a logical dividing point between client and server. Hence, the query and transac tion functionality related to SQL processing remained on the server side. In such an architecture, the server is often called a query server or transaction server because it provides these two functionalities. In an RDBMS, the server is also often called an SQL server.

The user interface programs and application programs can run on the client side. When DBMS access is required, the program establishes a connection to the DBMS (which is on the server side); once the connection is created, the client program can communicate with the DBMS. A standard called Open Database Connectivity (ODBC) provides an application programming interface (API), which allows client-side programs to call the DBMS, as long as both client and server machines have the necessary software installed. Most DBMS vendors provide ODBC drivers for their systems. A client program can actually connect to several RDBMSs and send query and transaction requests using the ODBC API, which are then processed at the server sites. Any query results are sent back to the client program, which can process and display the results as needed. A related standard for the Java programming language, called JDBC, has also been defined. This allows Java client programs to access one or more DBMSs through a standard interface. The architectures described here are called two-tier architectures because the software components are distributed over two systems: client and server. The advantages of this architecture are its simplicity and seamless compatibility with existing systems. The emergence of the Web changed the roles of clients and servers, leading to the three-tier architecture.

2.5.4 **Three-Tier and n-Tier Architectures**

for Web Applications Many Web applications use an architecture called the three-tier architecture, which adds an intermediate layer between the client and the database server, as illustrated in Figure 2.7(a). This intermediate layer or middle tier is called the application server or the Web server, depending on the application. This server plays an intermediary role by running application programs and storing business rules (procedures or constraints) that are used to access data from the database server. It can also improve database security by checking a client's credentials before forwarding a request to the database server. Clients contain user interfaces and Web browsers. The intermediate server accepts requests from the client, processes the request and sends database queries and commands to the database server, and then acts as a conduit for passing (partially) processed data from the database server to the clients, where it may be processed further and filtered to be presented to the users. Thus, the user interface, application rules, and data access act as the three tiers. Figure 2.7(b) shows another view of the three-tier architecture used by database and other application package vendors.

Centralized and Client/Server Architectures for DBMSs

- **Centralized DBMSs Architecture**
  - All DBMS functionality, application program execution, and user interface processing carried out on one machine

Basic Client/Server Architectures

- **Servers** with specific functionalities
  - **File server**
    - Maintains the files of the client machines.
  - **Printer server**
    - Connected to various printers; all print requests by the clients are forwarded to this machine
  - **Web servers** or **e-mail servers**

Basic Client/Server Architectures (cont'd.)

- **Client machines**
  - Provide user with:
    - Appropriate interfaces to utilize these servers
    - Local processing power to run local applications

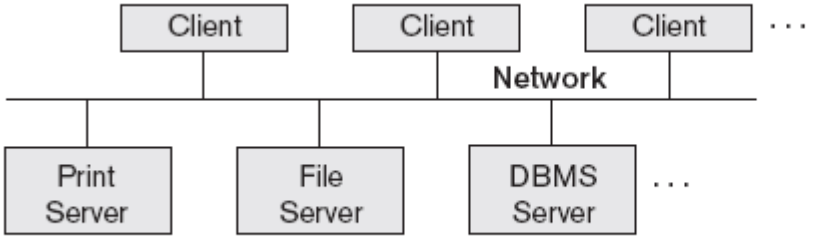


Figure 2.5
Logical two-tier client/server architecture.

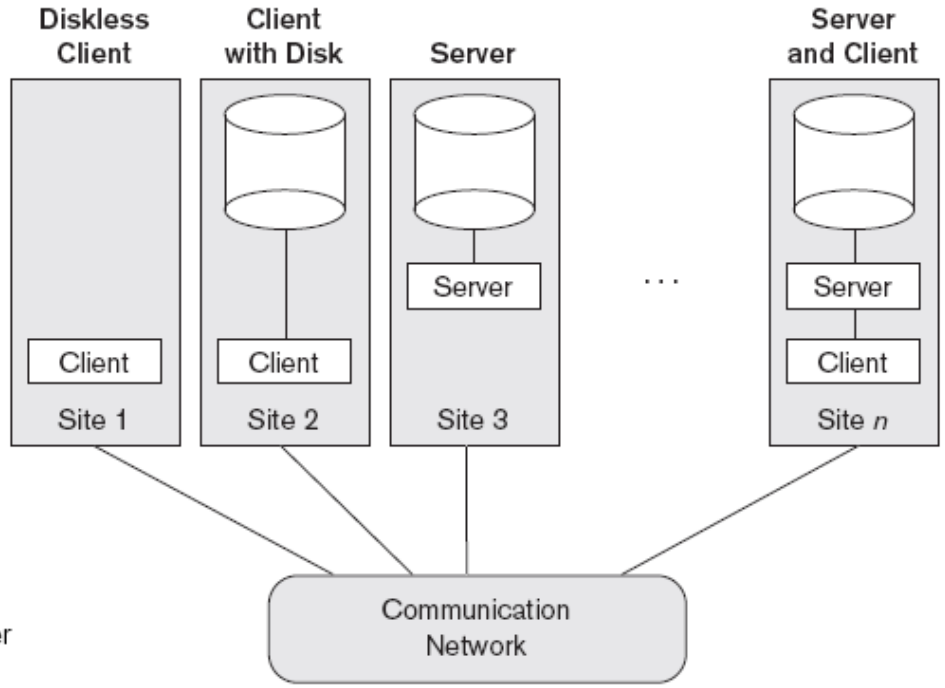


Figure 2.6
Physical two-tier client/server architecture.

Basic Client/Server Architectures (cont'd.)

- **Client**
  - User machine that provides user interface capabilities and local processing
- **Server**
  - System containing both hardware and software
  - Provides services to the client machines
    - Such as file access, printing, archiving, or database access

Two-Tier Client/Server Architectures for DBMSs

- Server handles
  - Query and transaction functionality related to SQL processing
- Client handles
  - User interface programs and application programs

Two-Tier Client/Server Architectures (cont'd.)

- Open Database Connectivity (ODBC)
  - Provides application programming interface (API)
  - Allows client-side programs to call the DBMS
    - Both client and server machines must have the necessary software installed
- JDBC
  - Allows Java client programs to access one or more DBMSs through a standard interface

Three-Tier and n-Tier Architectures for Web Applications

- **Application server** or **Web server**
  - Adds intermediate layer between client and the database server
  - Runs application programs and stores business rules
- N-tier
  - Divide the layers between the user and the stored data further into finer components

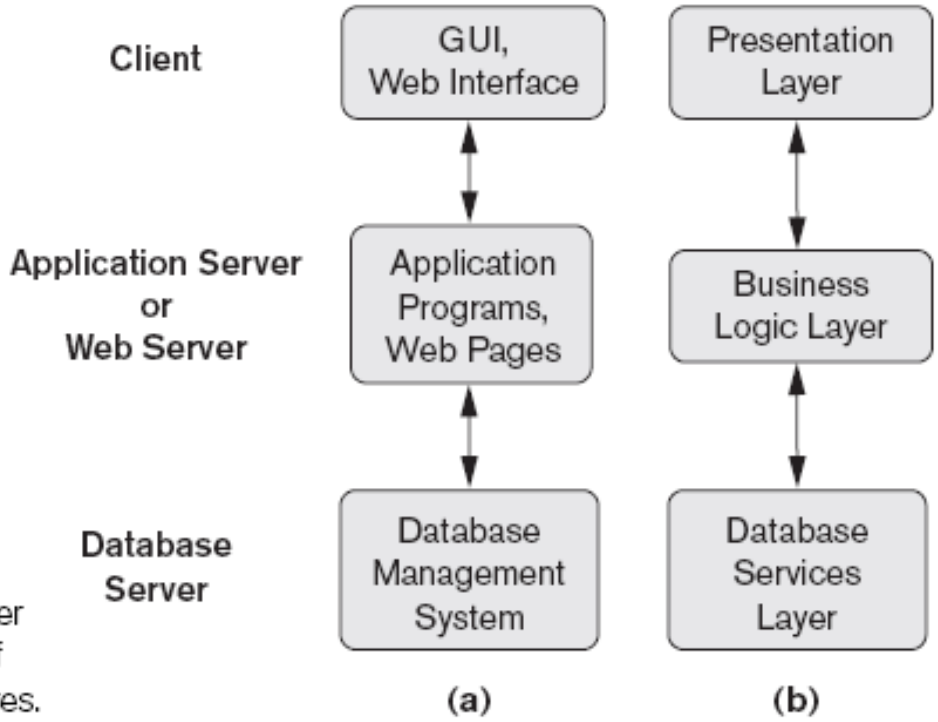


Figure 2.7
Logical three-tier client/server architecture, with a couple of commonly used nomenclatures.