Fundamentals of

Database
Systems

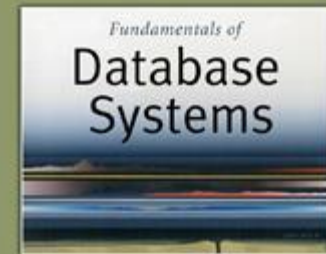**5**th Edition

Elmasri / Navathe

Chapter 2

Database System Concepts and Architecture

Outline

- Data Models and Their Categories
- History of Data Models
- Schemas, Instances, and States
- Three-Schema Architecture
- Data Independence
- DBMS Languages and Interfaces
- Database System Utilities and Tools
- Centralized and Client-Server Architectures
- Classification of DBMSs

Data Models

- **Data Model:**
  - A set of concepts to describe the *structure* of a database, the *operations* for manipulating these structures, and certain *constraints* that the database should obey.

- **Data Model Structure and Constraints:**
  - Constructs are used to define the database structure
  - Constructs typically include *elements* (and their *data types*) as well as groups of elements (e.g. *entity, record, table*), and *relationships* among such groups
  - Constraints specify some restrictions on valid data; these constraints must be enforced at all times

Data Models (continued)

- **Data Model Operations:**
  - These operations are used for specifying database *retrievals* and *updates* by referring to the constructs of the data model.
  - Operations on the data model may include *basic model operations* (e.g. generic insert, delete, update) and *user-defined operations* (e.g. compute_student_gpa, update_inventory)

Categories of Data Models

- **Conceptual (high-level, semantic) data models:**
  - Provide concepts that are close to the way many **users** perceive data.
    - (Also called *entity-based* or *object-based* data models.)
- **Physical (low-level, internal) data models:**
  - Provide concepts that describe details **of how data is stored in the computer.** These are usually specified in an ad-hoc manner through DBMS design and administration manuals
- **Implementation (representational) data models:**
  - Provide concepts that **fall between the above two**, used by many commercial DBMS implementations (e.g. relational data models used in many commercial systems).

Historical Development of Database Technology

- Early Database Applications:
  - The **Hierarchical and Network Models** were introduced in mid **1960s** and dominated during the seventies.
  - A bulk of the worldwide database processing still occurs using these models, particularly, the hierarchical model.
- Relational Model based Systems:
  - **Relational model** was originally introduced in **1970**, was heavily researched and experimented within **IBM** Research and several universities.
  - Relational DBMS **Products** emerged in the early **1980s**.

Historical Development of Database Technology (continued)

- Object-oriented and emerging applications:
  - **Object-Oriented** Database Management Systems (OODBMSs) were introduced in late 1980s and early 1990s to cater to the need of complex data processing in CAD and other applications.
    - Their use has not taken off much.
  - Many relational DBMSs have incorporated object database concepts, leading to a new category called *object-relational* DBMSs (ORDBMSs)
  - *Extended relational* systems add further capabilities (e.g. for multimedia data, XML, and other data types)

Extending Database Capabilities

- **New functionality is being added to DBMSs in the following areas**:
    - Scientific Applications
    - XML (eXtensible Markup Language)
    - Image Storage and Management
    - Audio and Video Data Management
    - Data Warehousing and Data Mining
    - Spatial Data Management
    - Time Series and Historical Data Management

- The above gives rise to *new research and development* in incorporating new data types, complex data structures, new operations and storage and indexing schemes in database systems.

- More **complex data structures** were needed for modeling the application than the simple relational representation.
  - **New data types** were needed in addition to the basic numeric and character string types.
  - **New operations and query language constructs** were necessary to manipulate the new data types.
  - **New storage and indexing structures** were needed for efficient searching on the new data types.

Schemas versus Instances

- Database Schema:
    - The ***description*** of a database.
    - Includes descriptions of the **database structure, data types, and the constraints** on the database.
- Schema Diagram:
    - An ***illustrative*** display of (most aspects of) a database schema.
- Schema Construct:
    - A ***component*** of the schema or an object within the schema, e.g., STUDENT, COURSE.

Schemas versus Instances

- Database State:
  - The **actual data** stored in a database at a *particular moment in time*. This includes the collection of all the data in the database.
  - Also called **database instance** (or **occurrence** or **snapshot**).
    - The term *instance* is also applied to individual database components, e.g. *record instance, table instance, entity instance*

Database Schema vs. Database State

- **Empty** Database State **(***with no data)*:
  - When we define a new database, we specify its database schema only to the DBMS**.**

- **Initial** Database State:
  - Refers to the database state when it is initially loaded with the initial data into the system.

- **Valid** State:
  - A state that satisfies the structure and constraints of the database**.**
  - DBMS **responsibility**: stores the descriptions of the schema constructs and constraints (**meta-data)**—in the DBMS catalog so that it can refer to the schema whenever it needs to.

Database Schema
vs. Database State (continued)

- Distinction
  - The *database schema* changes very infrequently.
  - The *database state* changes every time the database is updated.

- **Schema** is also called **intension**.
- **State** is also called **extension**.

Example of a Database Schema

Figure 2.1
Schema diagram for the database in Figure 1.2.

Example of a database state

COURSE

Course_name	Course_number	Credit_hours	Department
Intro to Computer Science	CS1310	4	CS
Data Structures	CS3320	4	CS
Discrete Mathematics	MATH2410	3	MATH
Database	CS3380	3	CS

SECTION

Section_identifier	Course_number	Semester	Year	Instructor
85	MATH2410	Fall	04	King
92	CS1310	Fall	04	Anderson
102	CS3320	Spring	05	Knuth
112	MATH2410	Fall	05	Chang
119	CS1310	Fall	05	Anderson
135	CS3380	Fall	05	Stone

GRADE_REPORT

Student_number	Section_identifier	Grade
17	112	B
17	119	C
8	85	A
8	92	A
8	102	B
8	135	A

PREREQUISITE

Course_number	Prerequisite_number
CS3380	CS3320
CS3380	MATH2410
CS3320	CS1310

Figure 1.2
A database that stores student and course information.
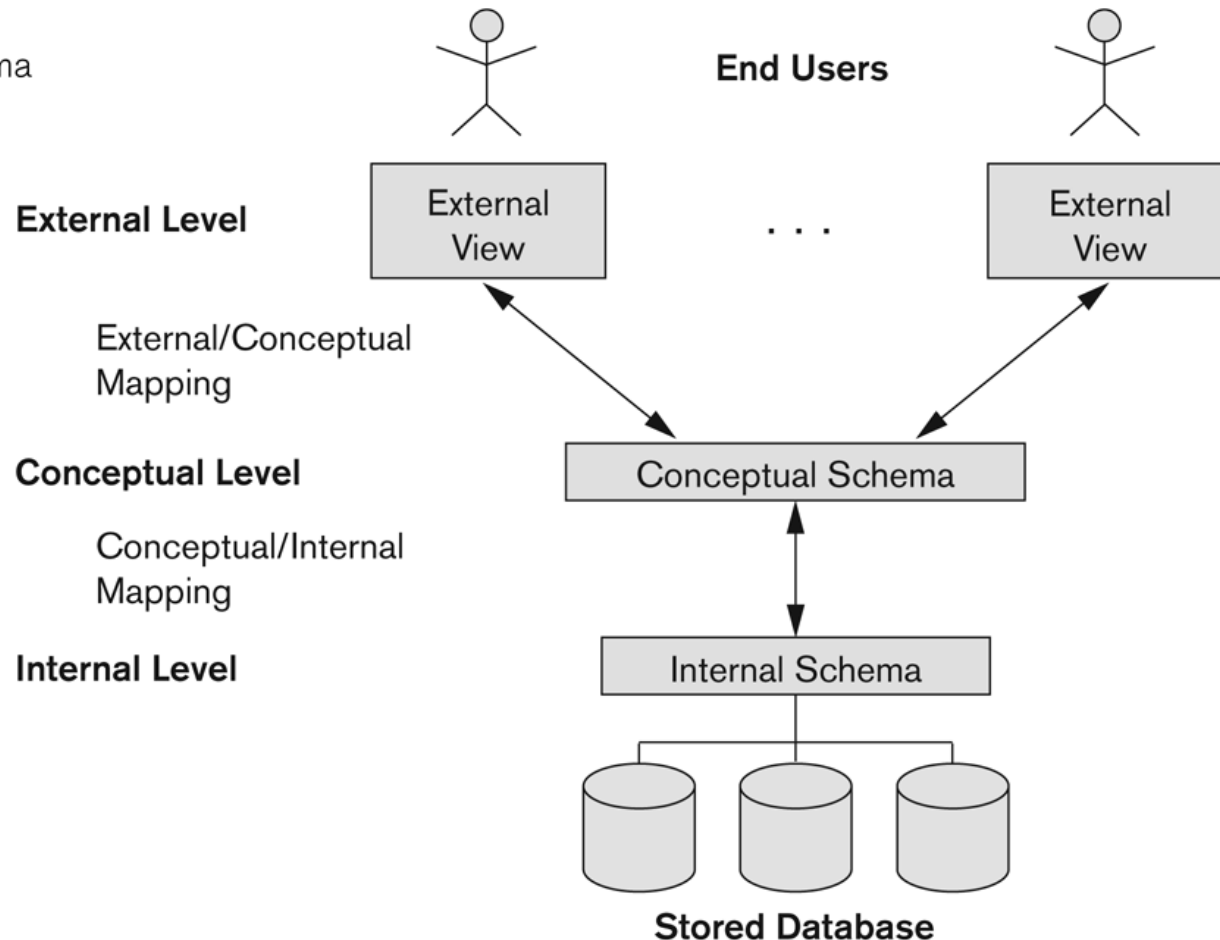
Three-Schema Architecture

- Proposed to support DBMS characteristics of:
    - **Program-data independence.**
    - Support of **multiple views** of the data.

- Not explicitly used in commercial DBMS products, but has been useful in explaining database system organization

- The goal of the three-schema architecture is to separate the user applications from the physical database

Three-Schema Architecture

- Defines DBMS schemas at *three* levels:
  - **Internal schema** at the internal level to describe **physical storage structures** and access paths (e.g indexes).
    - Typically uses a **physical** data model.
  - **Conceptual schema** at the conceptual level to describe the **structure and constraints** for the whole database for a community of **users**.
    - Uses a **conceptual** or an **implementation** data model.
  - **External schemas** at the external level to describe the **various user views**.
    - Usually uses the same data model as the conceptual schema.
    - 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.

The three-schema architecture

Figure 2.2
The three-schema architecture.

Three-Schema Architecture

- **Mappings** among schema levels are needed to **transform requests and data**.
  - Programs refer to an external schema, and are mapped by the DBMS to the internal schema for execution.
  - Data extracted from the internal DBMS level is reformatted to match the user's external view (e.g. formatting the results of an SQL query for display in a Web page)

Data Independence

- **Logical Data Independence:**
  - The capacity to change the conceptual schema without having to change the external schemas and their associated application programs.
- **Physical Data Independence:**
  - The capacity to change the internal schema without having to change the conceptual schema.
  - For example, the internal schema may be changed when certain file structures are reorganized or new indexes are created to improve database performance

Data Independence (continued)

- When a schema at a lower level is changed, only the **mappings** between this schema and higher-level schemas need to be changed in a DBMS that fully supports data independence.

- The higher-level schemas themselves are **unchanged**.

  - Hence, the application programs need not be changed since they refer to the external schemas.

DBMS Languages

- Data Definition Language (DDL)
- Data Manipulation Language (DML)
  - High-Level or Non-procedural Languages: These include the relational language SQL
    - May be used in a standalone way or may be embedded in a programming language
  - Low Level or Procedural Languages:
    - These must be embedded in a programming language

DBMS Languages

- **Data Definition Language (DDL):**
  - Used by the **DBA and database designers** to specify the conceptual schema of a database.
  - In many DBMSs, the DDL is also used to define internal and external schemas (views).
  - In some DBMSs, separate **storage definition language (SDL)** and **view definition language (VDL)** are used to define internal and external schemas.
    - SDL is typically realized via DBMS commands provided to the DBA and database designers

DBMS Languages

- **Data Manipulation Language (DML):**
  - Used to specify database retrievals and updates
  - DML commands (data sublanguage) can be *embedded* in a general-purpose programming language (host language), such as COBOL, C, C++, or Java.
    - A library of functions can also be provided to access the DBMS from a programming language
  - Alternatively, stand-alone DML commands can be applied directly (called a *query language*).

Types of DML

- **High Level or Non-procedural Language:**
    - For example, the SQL relational language
    - Are "set"-oriented and specify what data to retrieve rather than how to retrieve it.
    - Also called **declarative** languages.
- **Low Level or Procedural Language:**
    - Retrieve data one record-at-a-time;
    - Constructs such as looping are needed to retrieve multiple records, along with positioning pointers.

DBMS Interfaces

- **Stand-alone query language interfaces**
  - Example: Entering SQL queries at the DBMS interactive SQL interface (e.g. SQL*Plus in ORACLE)
- **Programmer interfaces for embedding DML in programming languages**
- **User-friendly interfaces**
  - Menu-based, forms-based, graphics-based, etc.
- **Interfaces for the DBA:**
  - Creating user accounts, granting authorizations
  - Setting system parameters
 - Changing schemas or access paths

DBMS Programming Language Interfaces

- Programmer interfaces for embedding DML in a programming languages:
    - **Embedded Approach**: e.g embedded SQL (for C, C++, etc.), SQLJ (for Java)
    - **Procedure Call Approach**: e.g. JDBC for Java, ODBC for other programming languages
    - **Database Programming Language Approach**: e.g. ORACLE has PL/SQL, a programming language based on SQL; language incorporates SQL and its data types as integral components

Database System Utilities

- To perform certain functions such as:
    - Loading data stored in files into a database. Includes data conversion tools.
    - **Backing** up the database periodically on tape.
    - Report generation utilities.
    - Performance monitoring utilities.
    - Other functions, such as sorting, user monitoring, data compression, etc.
    - Data dictionary / repository:
        - Used to store schema descriptions and other information such as design decisions, application program descriptions, user information, usage standards, etc

Typical DBMS Component Modules
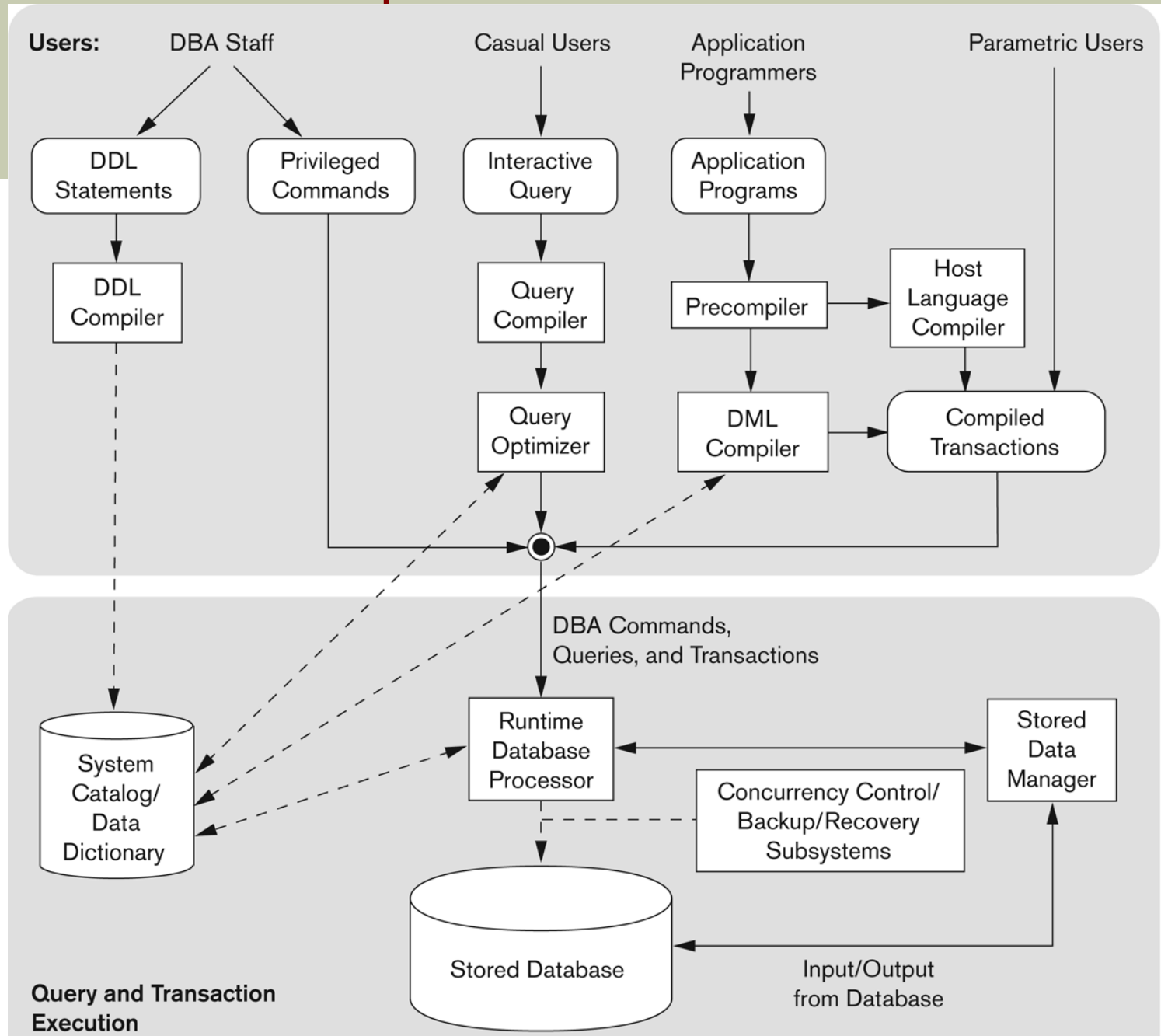
Centralized and Client-Server DBMS Architectures

- ## Centralized DBMS:

  - Combines everything into single system including- DBMS software, hardware, application programs, and user interface processing software.

  - User can still connect through a remote terminal – however, all processing is done at centralized site.
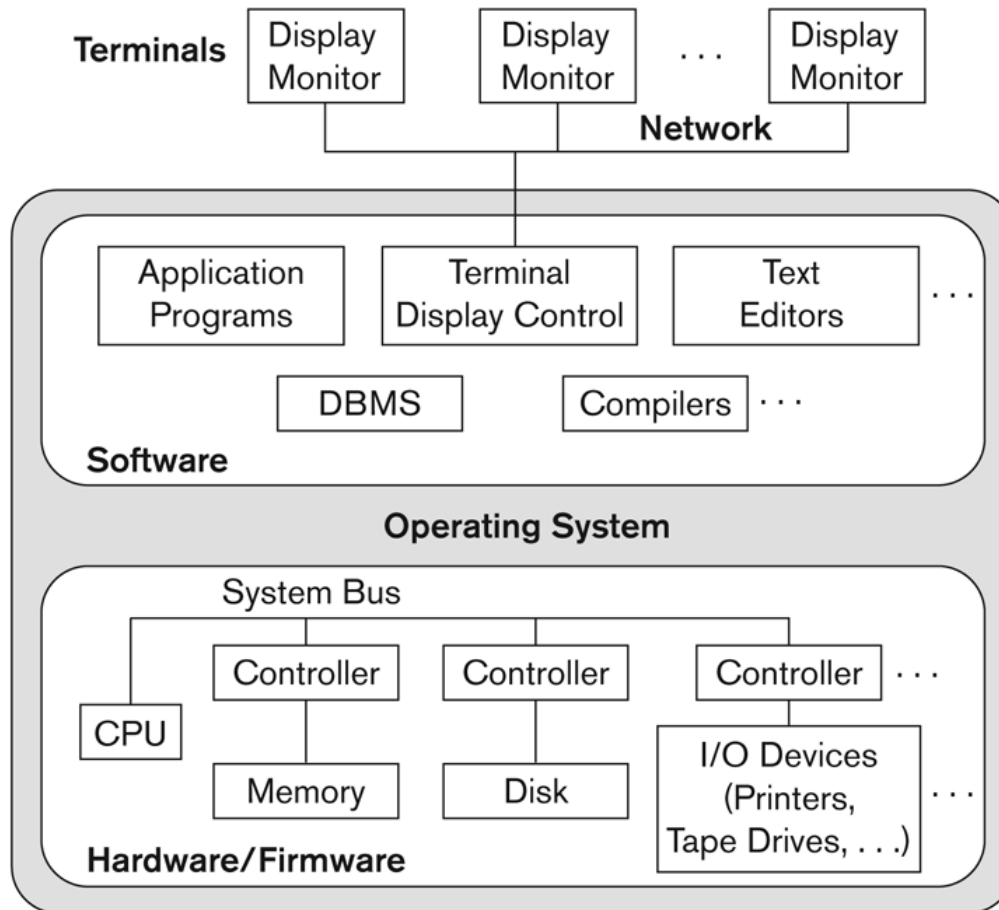
A Physical Centralized Architecture

Figure 2.4
A physical centralized architecture.

Basic 2-tier Client-Server Architectures

- Specialized Servers with Specialized functions
    - Print server
    - File server
    - DBMS server
    - Web server
    - Email server
- Clients can access the specialized servers as needed

Logical two-tier client server architecture

Figure 2.5
Logical two-tier client/server architecture.
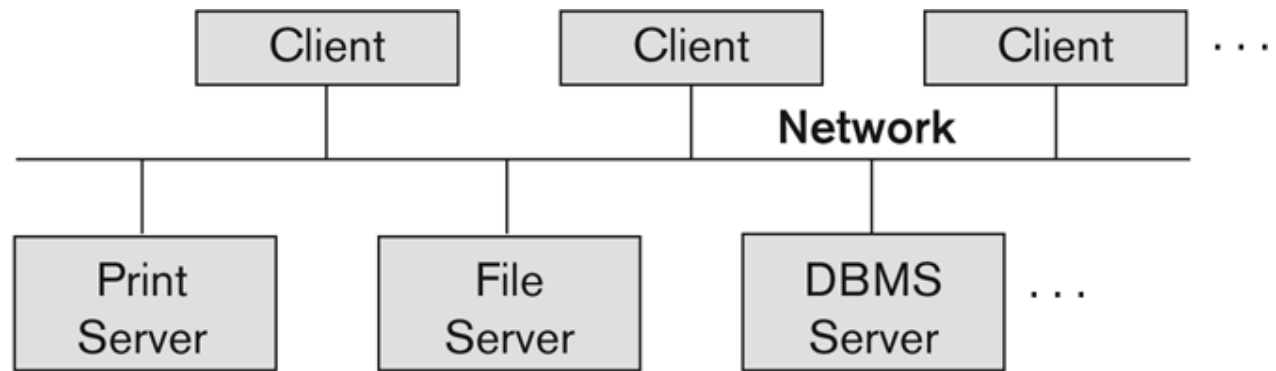
Clients

- Provide appropriate interfaces through a client software module to access and utilize the various server resources.

- Clients may be diskless machines or PCs or Workstations with disks with only the client software installed.

- Connected to the servers via some form of a network.
    - (LAN: local area network, wireless network, etc.)

DBMS Server

- Provides database query and transaction services to the clients
- Relational DBMS servers are often called SQL servers, query servers, or transaction servers
- Applications running on clients utilize an Application Program Interface (**API**) to access server databases via standard interface such as:
  - ODBC: Open Database Connectivity standard
  - JDBC: for Java programming access
- Client and server must install appropriate client module and server module software for ODBC or JDBC

Three Tier Client-Server Architecture

- Common for Web applications

- Intermediate Layer called **Application Server or Web Server:**
  - Stores the **web connectivity software and the business logic** part of the application used to access the corresponding data from the database server
  - Acts like a conduit for sending partially processed data between the database server and the client.

- Three-tier Architecture **Can Enhance Security**:
  - Database server only accessible via middle tier
  - Clients **cannot directly** access database server

Three-tier client-server architecture

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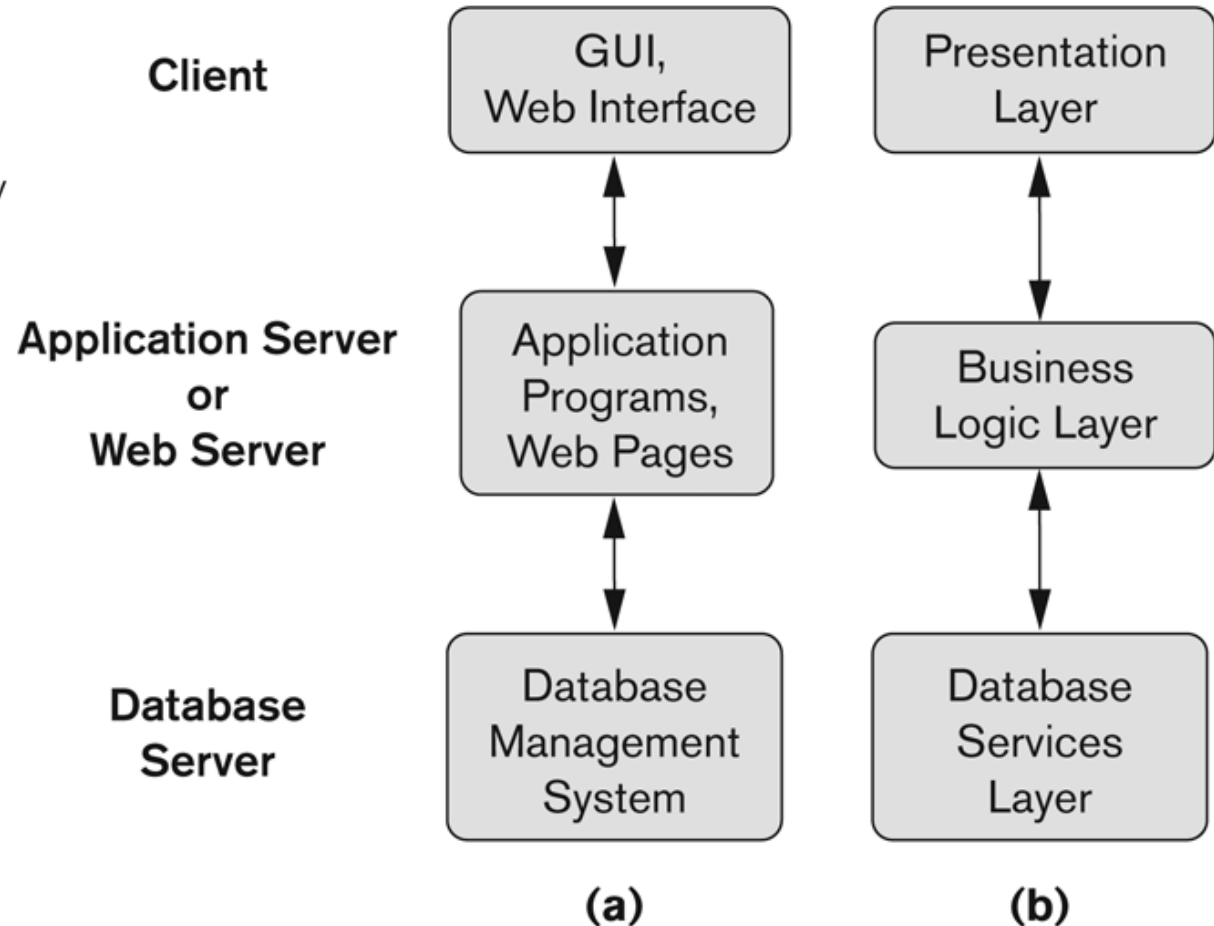
Classification of DBMSs

- Based on the data model used
    - Traditional: Relational, Network, Hierarchical.
    - Emerging: Object-oriented, Object-relational.
- Other classifications
    - Single-user (typically used with personal computers)
      vs. multi-user (most DBMSs).
    - Centralized (uses a single computer with one database)
      vs. distributed (uses multiple computers, multiple databases)

Variations of Distributed DBMSs (DDBMSs)

- Homogeneous DDBMS

- Heterogeneous DDBMS

- Distributed Database Systems have now come to be known as client-server based database systems because:

  - They do not support a totally distributed environment, but rather a set of database servers supporting a set of clients.

Cost considerations for DBMSs

- Cost Range: from free open-source systems to configurations costing millions of dollars
- Examples of free relational DBMSs: MySQL, PostgreSQL, others
- Commercial DBMS offer additional specialized modules, e.g. time-series module, spatial data module, document module, XML module
  - These offer additional specialized functionality when purchased separately
  - Sometimes called cartridges (e.g., in Oracle) or blades
- Different licensing options: site license, maximum number of concurrent users (seat license), single user, etc.

Summary

- Data Models and Their Categories
- History of Data Models
- Schemas, Instances, and States
- Three-Schema Architecture
- Data Independence
- DBMS Languages and Interfaces
- Database System Utilities and Tools
- Centralized and Client-Server Architectures
- Classification of DBMSs