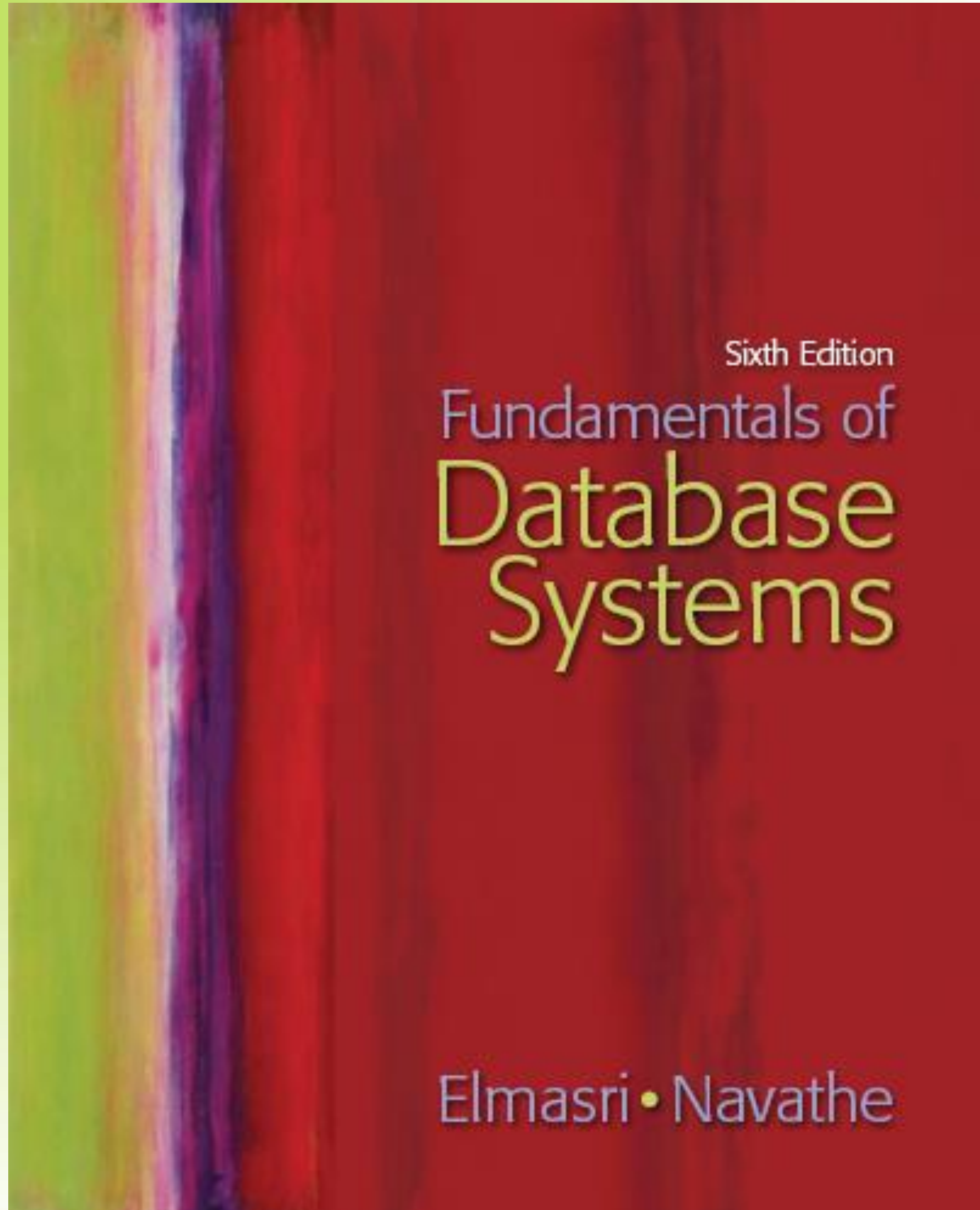
Chapter 2

Database System Concepts and Architecture



Chapter 2

Database System Concepts and Architecture

Chapter 2 Outline

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

Data Models

- **Data Model:**
  - A set of concepts to describe the ***structure*** of a database, the ***operations*** for manipulating the data, and the ***constraints*** that the data should follow.
- **Data Model Structure and Constraints:**
  - Data Model **constructs** define the database structure
  - Data model constructs often include: ***data elements*** and their ***data types*** (often called **attributes**); grouping of related elements into ***entities*** (also called **objects** or **records** or **tuples**); and ***relationships*** among entities
  - Constraints specify ***restrictions*** on the stored data; the data that satisfies the constraints is called ***valid data***

Addison-Wesley is an imprint of

PEARSON

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).

Database Schema versus Database State

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

Database Schema vs. Database State (cont.)

- 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 a database instance (or occurrence or snapshot).
    - NOTE: The term *instance* is also used to refer to individual database components, e.g. a *record instance, table instance,* or *entity instance*

Addison-Wesley is an imprint of PEARSON

Database Schema vs. Database State

- Database State:
  - Refers to the ***content*** of a database at a particular moment in time.
- Initial Database State:
  - Refers to the database state when it is initially loaded into the system.
- Valid State:
  - A state that satisfies the structure and constraints of the database.

Database Schema vs. Database State (cont.)

- 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**.

Addison-Wesley is an imprint of PEARSON

Example of a Database Schema

STUDENT

Name	Student_number	Class	Major

COURSE

Course_name	Course_number	Credit_hours	Department

PREREQUISITE

Course_number	Prerequisite_number

SECTION

Section_identifier	Course_number	Semester	Year	Instructor

GRADE_REPORT

Student_number	Section_identifier	Grade

Figure 2.1
Schema diagram for the database in Figure 1.2.

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

Addison-Wesley is an imprint of PEARSON

Three-Schema Architecture (cont.)

- 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 an **implementation** (or a conceptual) data model.
  - **External schemas** at the external level to describe the various user views.
    - Usually uses the same data model as the conceptual schema.

The three-schema architecture (cont.)

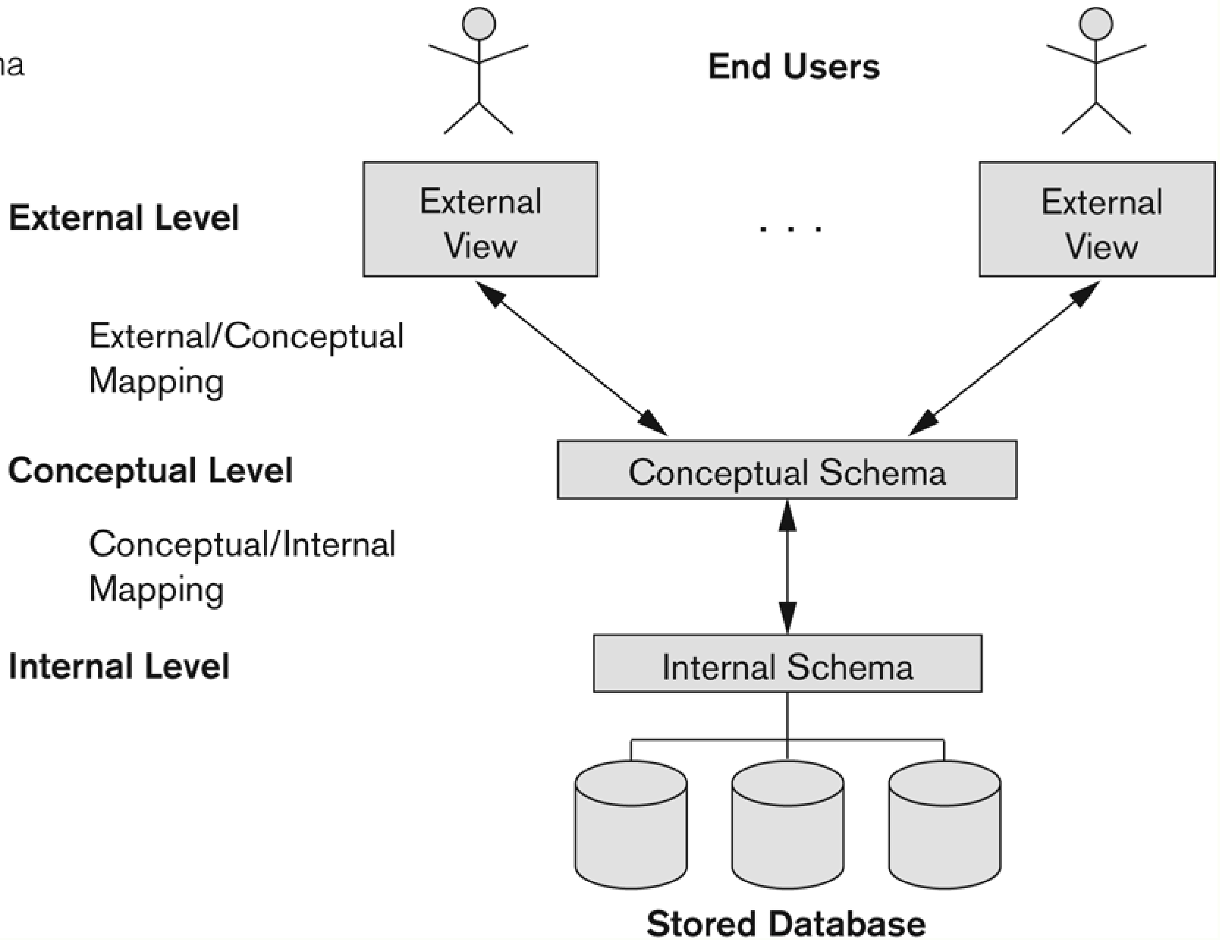


Figure 2.2
The three-schema architecture.

Addison-Wesley is an imprint of PEARSON

Three-Schema Architecture (cont.)

- Mappings among schema levels are needed to transform requests and data.
  - Users and 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 as 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

Addison-Wesley is an imprint of

PEARSON

Data Independence (cont.)

- 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):**
  - 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).
  - Theoretically, separate **storage definition language (SDL)** and **view definition language (VDL)** can used to define internal and external schemas. In practice:
    - SDL is typically realized via DBMS commands provided to the DBA and database designers
    - VDL is typically part of the same language as DDL

Addison-Wesley
is an imprint of
PEARSON

DBMS Languages (cont.)

- **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 (see Chapter 13)
    - 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*).

DBMS Languages (cont.)

- Types of DML:
  - High-Level Declarative (Set-oriented, Non-procedural) Languages, such as the relational language SQL
    - Specify “what” data to retrieve rather then “how” to retrieve it
    - May be used in a standalone way or may be embedded in a programming language
  - Low Level or Procedural (Record-at-a-time) Languages:
    - Must be embedded in a programming language
    - Need programming language constructs such as looping

DBMS Interfaces

- Stand-alone query language interfaces
  - Example: Typing SQL queries directly through the DBMS interactive SQL interface (e.g. SQL*Plus in ORACLE)
- Programmer interfaces for embedding DML in programming languages
- User-friendly interfaces (often Web-based)
  - Menu-based, forms-based, graphics-based, etc.

DBMS Programming Language Interfaces

- Programmer interfaces for embedding DML in a programming language (see Chapter 13):
  - **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

User-Friendly and Web-based DBMS Interfaces

- Menu-based, popular for browsing on the web
- Forms-based, designed for naïve users
- Graphics-based
  - (Point and Click, Drag and Drop, etc.)
- Natural language: requests in written English
- Combinations of the above:
  - For example, both menus and forms used extensively in Web database interfaces

Other DBMS Interfaces

- Speech as Input and Output
- Web Browser as an interface
- Parametric interfaces, e.g., bank tellers using function keys.
- Interfaces for the DBA:
  - Creating user accounts, granting authorizations
  - Setting system parameters
  - Changing schemas or storage structures/access paths (physical database)

Addison-Wesley is an imprint of PEARSON

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.
  - Reorganizing database file structures.
  - Report generation utilities.
  - Performance monitoring utilities.
  - Other functions, such as sorting, user monitoring, data compression, etc.

Other Tools

- Data dictionary/repository:
  - Used to store schema descriptions and other information such as design decisions, application program descriptions, user information, usage standards, etc.
  - **Active data dictionary** is accessed by DBMS software and users/DBA.
  - **Passive data dictionary** is accessed by users/DBA only.

Other Tools (cont.)

- Application Development Environments and CASE (computer-aided software engineering) tools often have a database design component
- Examples:
  - PowerBuilder (Sybase)
  - JBuilder (Borland)
  - JDeveloper 10G (Oracle)

Typical DBMS Component Modules

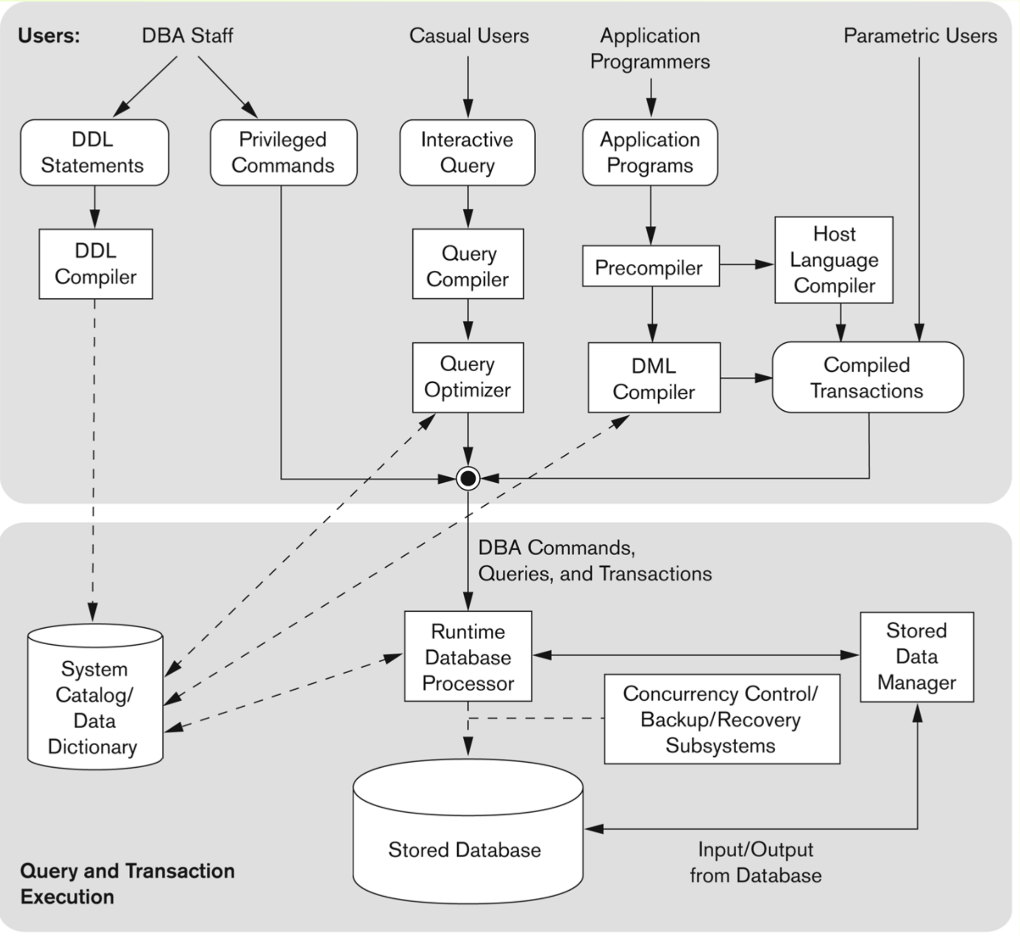


Figure 2.3
Component modules of a DBMS and their interactions.

Addison-Wesley is an imprint of PEARSON

DBMS Architectures

- Centralized DBMS Architecture:
  - Combines everything into single computer 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 (computer).

A Physical Centralized Architecture

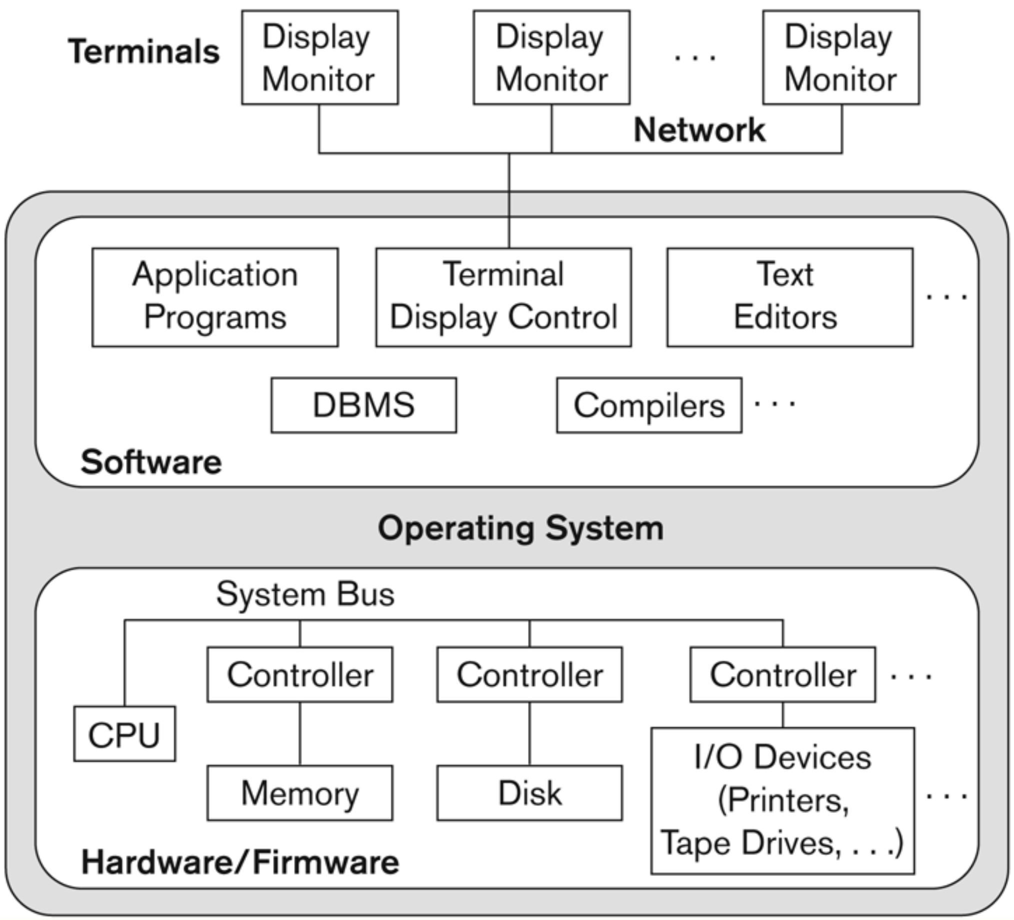


Figure 2.4
A physical centralized architecture.

DBMS Architectures (cont.)

- Basic 2-tier Client-Server Architecture: Specialized Server nodes with Specialized functions
  - Print server
  - File server
  - DBMS server
  - Web server
  - Email server
- Client nodes can access the specialized servers as needed

Logical two-tier client server architecture

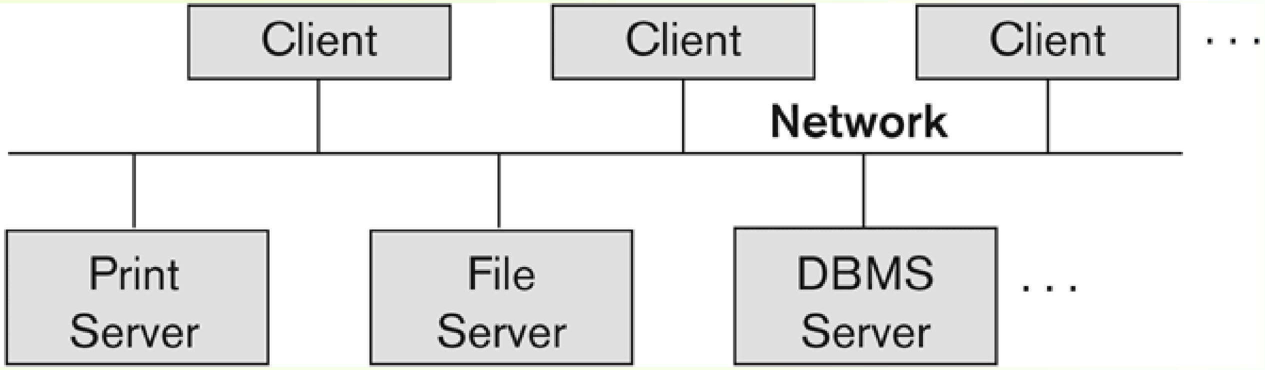


Figure 2.5
Logical two-tier client/server architecture.

Client nodes

- Provide appropriate interfaces through a client software module to access and utilize the various server resources.
- Clients may be PCs or Workstations (or even diskless machines) with the client software installed.
- Connected to the servers via 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
- See Chapter 13

Two Tier Client-Server DBMS Architecture

- A program running at a client may connect to several DBMSs (also called data sources).
- In general, data sources can be files or other non-DBMS software that manages data.
- Client focuses on user interface interactions and only accesses database when needed.
- In some cases (e.g. some object DBMSs), more functionality is transferred to clients (e.g. data dictionary functions, optimization and recovery across multiple servers, etc.)

Three Tier Client-Server DBMS Architecture

- Common for Web applications
- Third intermediate layer (middle tier) called Application Server or Web Server:
  - Stores the web connectivity software and the business logic part of the application
  - Accesses and updates data on 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

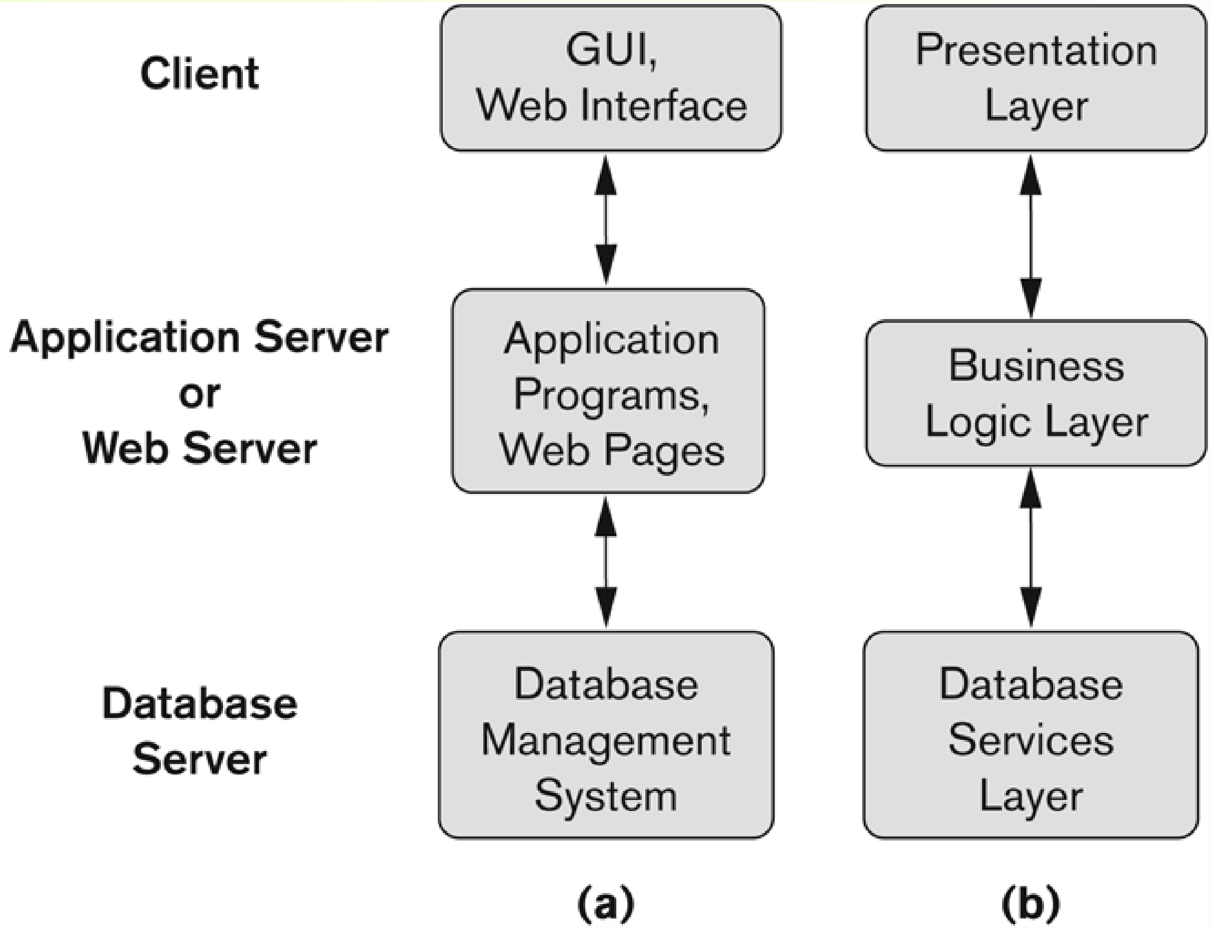


Figure 2.7
Logical three-tier client/server architecture, with a couple of commonly used nomenclatures.

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, see Chapter 25)

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 DBMSs 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.

History of Data Models

- Network Model
- Hierarchical Model
- Relational Model
- Object-oriented Data Models
- Object-Relational Models

Addison-Wesley
is an imprint of
PEARSON

History of Data Models (cont.)

- **Network Model:**
  - The first network DBMS was implemented by Honeywell in 1964-65 (IDS System).
  - Adopted heavily due to the standard support by CODASYL (Conference on Data Systems Languages) (CODASYL - DBTG report of 1971).
  - Later implemented in a large variety of systems - IDMS (Cullinet - now Computer Associates), DMS 1100 (Unisys), IMAGE (H.P. (Hewlett-Packard)), VAX -DBMS (Digital Equipment Corp., next COMPAQ, now H.P.).

Example of Network Model Schema

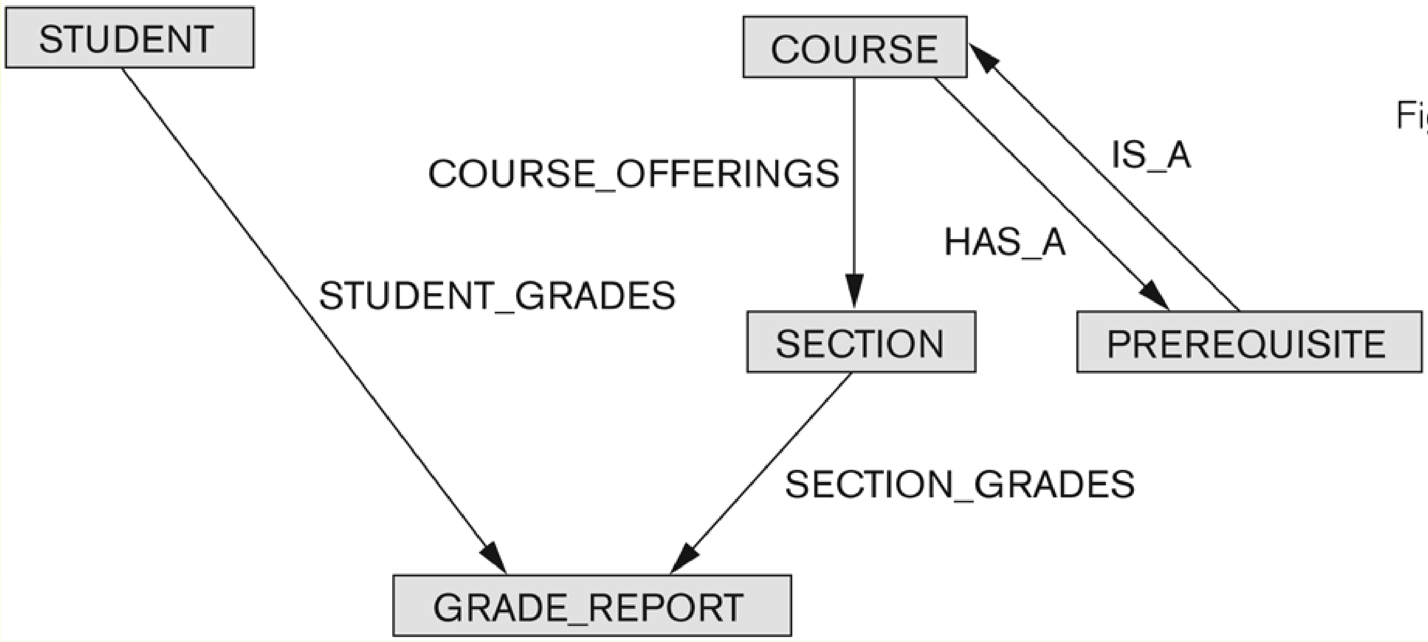


Figure 2.8
The schema of Figure 2.1 in network model notation.

Network Model

- Advantages:
  - Can model complex relationships among records and represents semantics of add/delete on the relationships.
  - Can handle most situations for modeling using record types and relationship types.
  - Language is navigational; uses constructs like FIND, FIND member, FIND owner, FIND NEXT within set, GET, etc.
    - Programmers can do optimal navigation through the database.

Addison-Wesley is an imprint of PEARSON

Network Model

- Disadvantages:
  - Navigational and procedural nature of processing requires programming access
  - Intermixes storage structures with conceptual modeling relationships
  - Database contains a complex array of pointers that thread through a set of records.
    - Little scope for automated “query optimization”

Addison-Wesley is an imprint of PEARSON

History of Data Models (cont.)

- **Hierarchical Data Model:**
  - Initially implemented in a joint effort by IBM and North American Rockwell around 1965. Resulted in the IMS family of systems.
  - IBM's IMS product had a very large customer base worldwide
  - Hierarchical model was formalized based on the IMS system
  - Other systems based on this model: System 2k (SAS inc.)

Hierarchical Model

- Advantages:
  - Can implement certain tasks very efficiently
  - Easy to store hierarchically organized data, e.g., organization (“org”) charts
- Disadvantages:
  - Navigational and procedural nature of processing
  - Difficult to store databases where multiple relationships exist among the data records
  - Little scope for “query optimization” by system (programmer must optimize the programs)
  - Language is procedural: Uses constructs like GET, GET UNIQUE, GET NEXT, GET NEXT WITHIN PARENT, etc.

History of Data Models (cont.)

- **Relational Model:**
  - Proposed in 1970 by E.F. Codd (IBM), first commercial systems in early 1980s.
  - Now in many commercial products (e.g. DB2, ORACLE, MS SQL Server, SYBASE, INFORMIX).
  - Several free open source implementations, e.g. MySQL, PostgreSQL
  - Currently most dominant for developing database applications.
  - SQL relational standards: SQL-89 (SQL1), SQL-92 (SQL2), SQL-99, SQL3, SQL-2008
  - Chapters 3 through 6 describe this model in detail

History of Data Models (cont.)

- **Object-oriented Data Models:**
  - Allow databases to be used seamlessly with object-oriented programming languages.
  - Can store persistent objects created in O-O Programming Languages such as C++ (e.g., in OBJECTSTORE or VERSANT), and Smalltalk (e.g., in GEMSTONE).
  - Other experimental systems include O2, ORION (at MCC - then ITASCA), IRIS (at H.P.- used in Open OODB).
  - Object Database Standard: ODMG-93, ODMG-version 2.0, ODMG-version 3.0.
  - Chapter 11 describes this model.

History of Data Models (cont.)

- **Object-Relational Models:**
  - Relational systems incorporated concepts from object databases leading to object-relational.
  - Exemplified in the latest versions of Oracle-10i, DB2, and SQL Server and other DBMSs.
  - Standards started in SQL-99 and enhanced in SQL-2008.
  - Chapter 11 also describes this model.

Summary

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