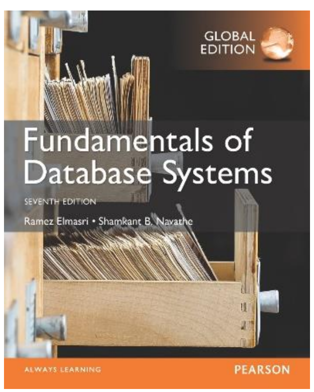
Database Systems

Department of Computer Science
Chungbuk National University

Jong Yun LEE

CHAPTER 2:
Database System Concepts and Architecture



Introduction

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, Schemas, and Instances

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).

- **Self-Describing Data Models:**
  - Combine the description of data with the data values. Examples include XML, key-value stores and some NOSQL systems.

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.

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

Schema diagram for the database

STUDENT

Name	Student_number	Class	Major
Smith	17	1	CS
Brown	8	2	CS

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	07	King
92	CS1310	Fall	07	Anderson
102	CS3320	Spring	08	Knuth
112	MATH2410	Fall	08	Chang
119	CS1310	Fall	08	Anderson
135	CS3380	Fall	08	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

A database that stores student and course information

Schemas versus Instances

- **Schema Construct:**
  - A ***component*** of the schema or an object within the schema, e.g., `STUDENT`, `COURSE`.

- **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

- **Database State:**
  - Refers to the ***content*** of a database at a 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
(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

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.

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 and Data Independence

Three-Schema Architecture

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

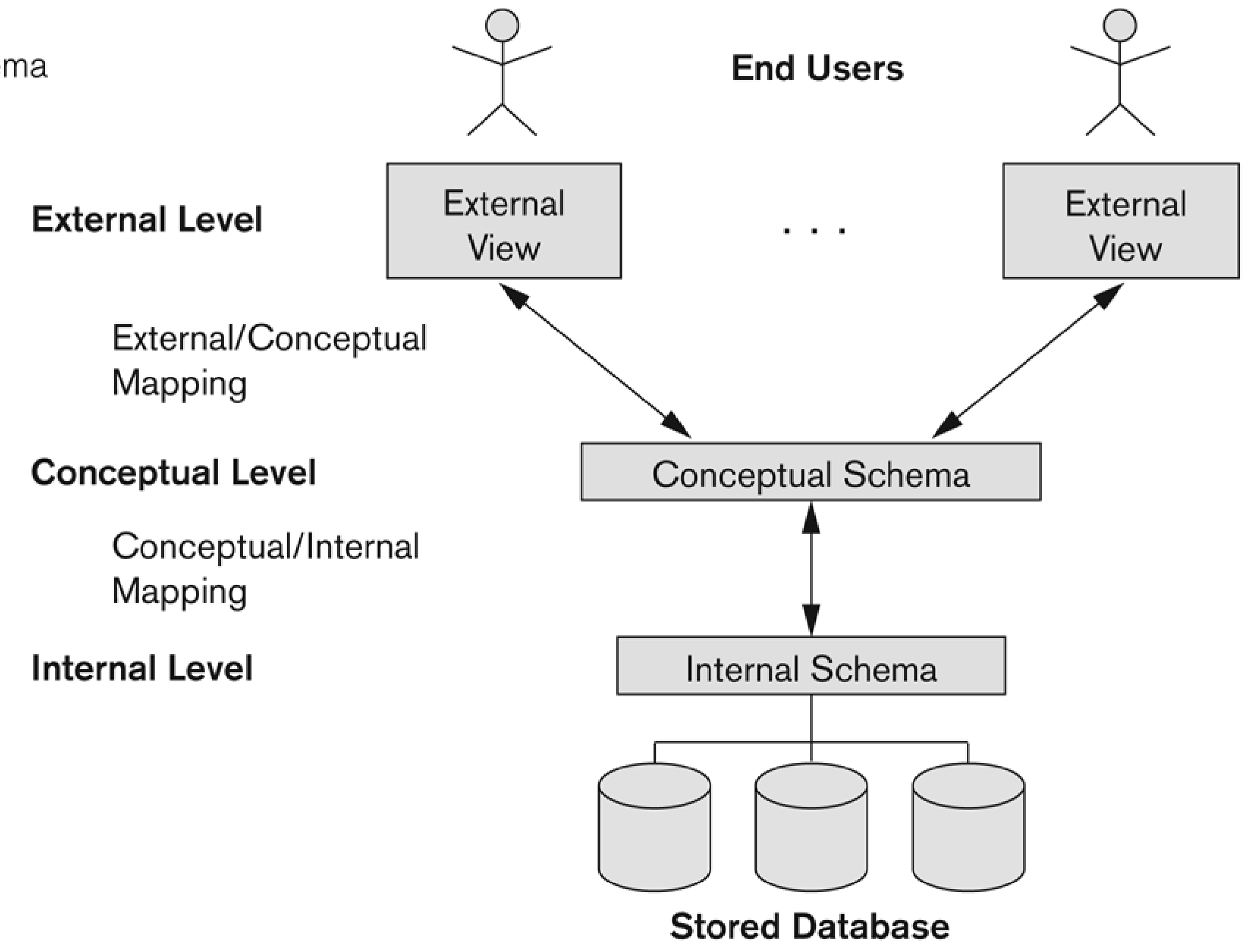
- **Not explicitly used in commercial DBMS products, but has been useful in explaining database system organization**

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.

Three-Schema Architecture

Figure 2.2
The three-schema architecture.



Three-Schema Architecture

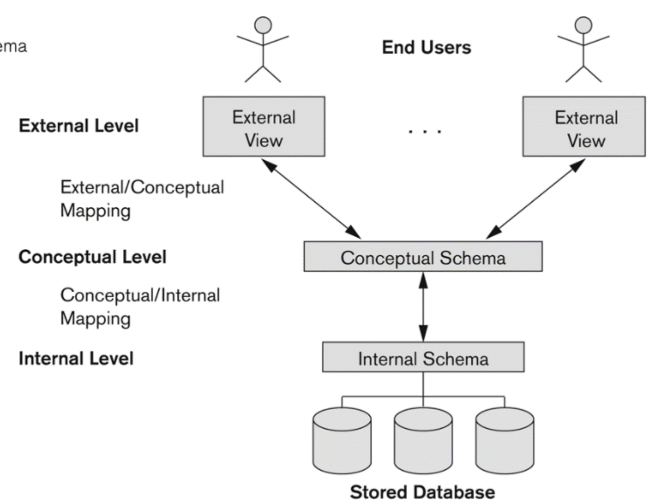
- **Mappings among schema levels are <u>needed to transform requests and data</u>.**
  - Programs refer to an external schema, and are mapped by the DBMS to the internal schema for execution.
  - Data extracted from the internal DBMS level is reformatted to match the user's external view (e.g. formatting 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

Figure 2.2
The three-schema architecture.



Data Independence (continued)

- **When a schema at a lower level is changed, <u>only the mappings</u> between this schema and higher-level schemas <u>need to be changed in a DBMS</u> 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.

Database Languages and Interfaces

DBMS Languages

- **Data Definition Language (DDL)**

- **Data Manipulation Language (DML)**
  - High-Level or Non-procedural Languages: These include the relational language SQL: (Interactive Query Language)
    - May be <u>used in a standalone way</u> or may be <u>embedded in a programming language</u>
  - Low Level or Procedural Languages: (Embedded Language)
    - These must be embedded in a programming language

DBMS Languages

- **Data Definition Language (DDL):**
  - Used by the DBA and database designers to <u>specify the conceptual schema of a database.</u>
  - 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 <u>database retrievals and updates</u>
  - 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:**

  iterator

  - 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.
- **Mobile Interfaces: interfaces allowing users to perform transactions using mobile apps**

DBMS Programming Language Interfaces

- **Embedded Approach: e.g embedded SQL (for C, C++, etc.), SQLJ (for Java)**

- **Procedure Call Approach: e.g. JDBC for Java, ODBC (Open Database Connectivity) for other programming languages as API's (application programming interfaces)**

Multi-row query

JDBC

```
PreparedStatement stmt = conn.prepareStatement(
   "SELECT LASTNAME"
 + " , FIRSTNME"
 + " , SALARY"
 + " FROM DSN8710.EMP"
 + " WHERE SALARY BETWEEN ? AND ?");
stmt.setBigDecimal(1, min);
stmt.setBigDecimal(2, max);
ResultSet rs = stmt.executeQuery();
while (rs.next()) {
  lastname = rs.getString(1);
  firstname = rs.getString(2);
  salary = rs.getBigDecimal(3);
  // Print row...
}
rs.close();
stmt.close();
```

SQLJ

```
#sql private static iterator EmployeeIterator(String, String,
BigDecimal);
...
EmployeeIterator iter;
#sql [ctx] iter = {
  SELECT LASTNAME
       , FIRSTNME
       , SALARY
    FROM DSN8710.EMP
   WHERE SALARY BETWEEN :min AND :max
};
do {
  #sql {
    FETCH :iter
     INTO :lastname, :firstname, :salary
  };
  // Print row...
} while (!iter.endFetch());
iter.close();
```

DBMS Programming Language Interfaces

- **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**

- **Scripting Languages: PHP (client-side scripting) and Python (server-side scripting) are used to write database programs.**

```
<<label>>   -- this is optional
DECLARE
-- this section is optional
  number1 NUMBER(2);
  number2 number1%TYPE := 17;             -- value default
  text1   VARCHAR2(12) := 'Hello world';
  text2   DATE         := SYSDATE;        -- current date and time
BEGIN
-- this section is mandatory, must contain at least one executable statement
  SELECT street_number
    INTO number1
    FROM address
    WHERE name = 'INU';
EXCEPTION
-- this section is optional
   WHEN OTHERS THEN
     DBMS_OUTPUT.PUT_LINE('Error Code is ' || TO_CHAR(sqlcode));
     DBMS_OUTPUT.PUT_LINE('Error Message is ' || sqlerrm);
END;
```

User-Friendly DBMS Interfaces

- **Menu-based (Web-based), popular for browsing on the web**
- **Forms-based, designed for naïve users used to filling in entries on a form**
- **Graphics-based**
  - Point and Click, Drag and Drop, etc.
  - Specifying a query on a schema diagram
- **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

- **Natural language: free text as a query**
- **Speech : Input query and Output response**
- **Web Browser with keyword search**
- **Parametric interfaces, e.g., bank tellers using function keys.**
- **Interfaces for the DBA:**
  - Creating user accounts, granting authorizations
  - Setting system parameters
  - Changing schemas or access paths

The Database System Environment

Typical DBMS Component Modules

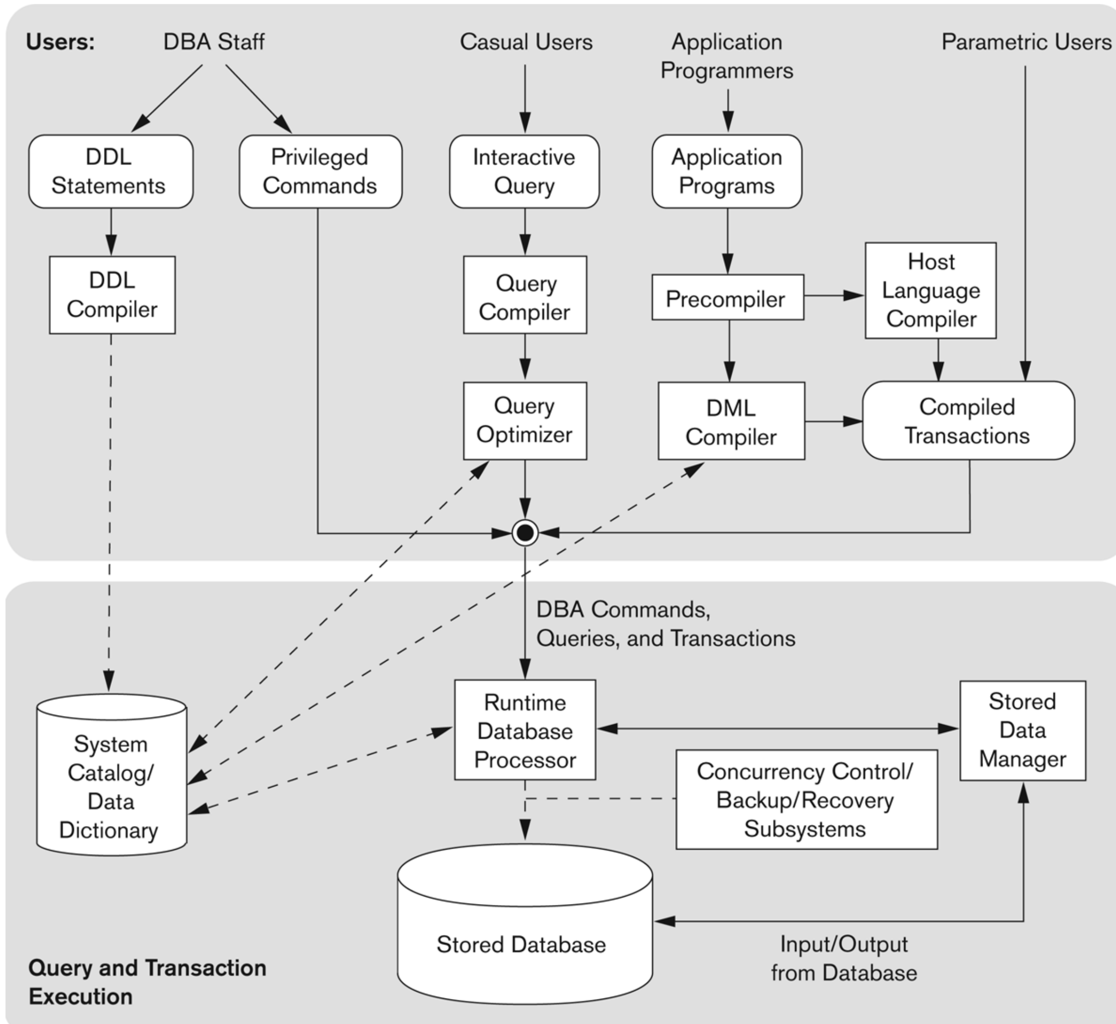


Figure 2.3
Component modules of a DBMS and their interactions.

Runtime Database
Query Processor

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.
  - Performance monitoring utilities.
  - Report generation 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

- **Application Development Environments and CASE (computer-aided software engineering) tools:**
- **Examples:**
  - PowerBuilder (Sybase)
  - JBuilder (Borland)
  - JDeveloper 10G (Oracle)

Centralized and Client/Server Architectures for DBMS

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

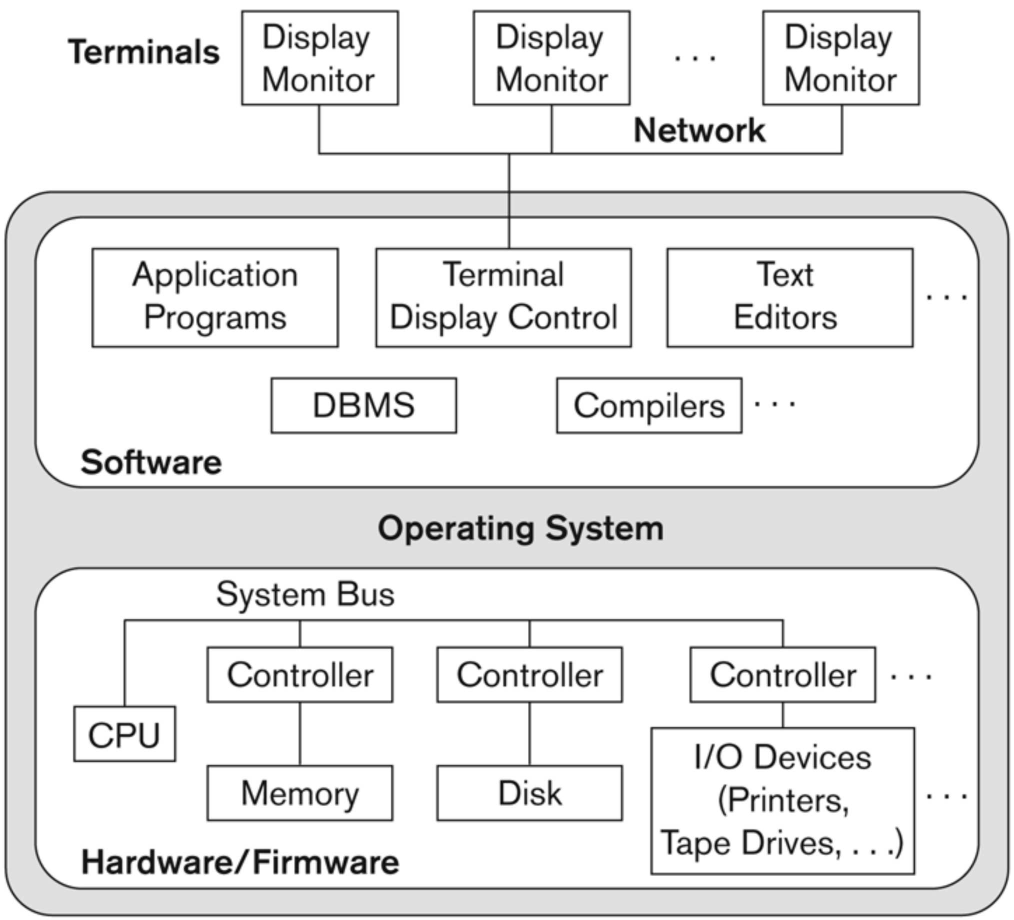


Figure 2.4
A physical centralized architecture.

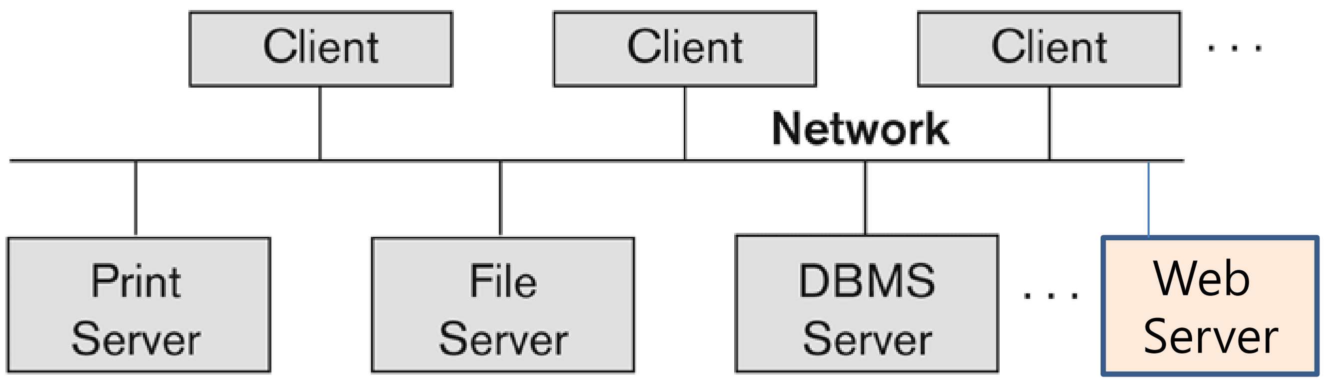
Basic 2-tier Client-Server Architectures

- **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

Two Tier Client-Server Architecture

- **Client and server must install appropriate client module and server module software for ODBC or JDBC**

- **A client program may connect to several DBMSs, sometimes called the data sources.**

- **In general, data sources can be files or other non-DBMS software that manages data.**

- **See Chapter 10 for details on Database Programming**

Three Tier Client-Server Architecture

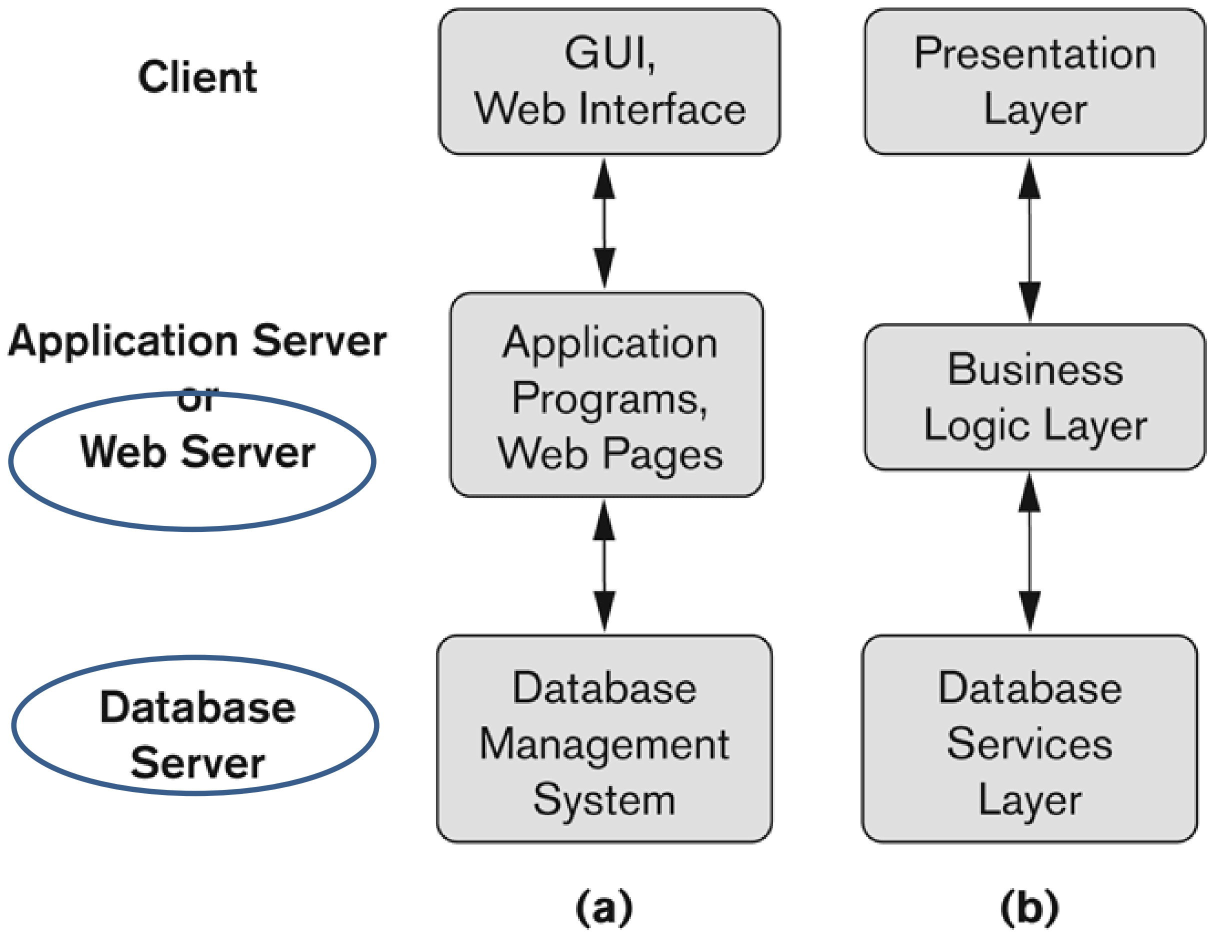
- **Common for Web applications**

- **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
  - Clients contain user interfaces and Web browsers
  - The client is typically a PC or a mobile device connected to the Web

Three-tier client-server architecture

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



Classification of Database Management Systems

Classification of DBMSs

- **Based on the data model used**
  - Legacy: Network, Hierarchical.
  - Currently Used: **Relational**, **Object-oriented**, Object-relational
  - Recent Technologies: Key-value storage systems, **NOSQL systems**: document based, column-based, graph-based and key-value based. Native XML DBMSs.

- **Other classifications**
  - Single-user (typically used with personal computers) vs. multi-user (most DBMSs).
  - Centralized (uses a single computer with one database) vs. distributed (multiple computers, multiple DBs)

Variations of Distributed DBMSs (DDBMSs)

- **Homogeneous DDBMS**
- **Heterogeneous DDBMS**
- **Federated or Multidatabase Systems**
  - Participating Databases are loosely coupled with high degree of autonomy.
- **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.**

Other Considerations

- **Type of access paths within database system**
  - E.g.- inverted indexing based (ADABAS is one such system).Fully indexed databases provide access by any keyword (used in search engines)

- **General Purpose vs. Special Purpose**
  - E.g.- Airline Reservation systems or many others-reservation systems for hotel/car etc. Are special purpose OLTP (Online Transaction Processing Systems)

History of Data Model

History of Data Models (Additional Material)

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

Network Model

- **The first network DBMS was implemented by Honeywell in 1964-65 (IDS System).**

- **Adopted heavily due to the 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.).**

Network Model

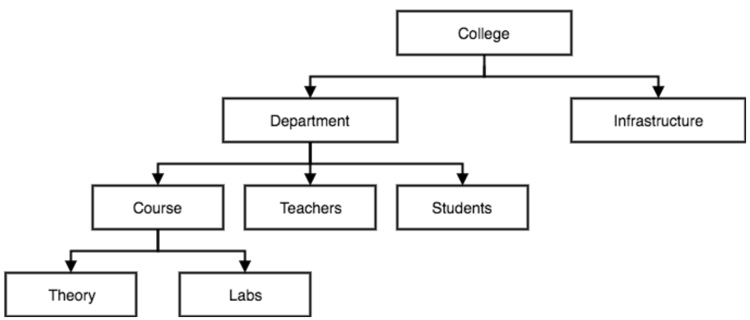
- **Advantages:**
  - Network Model is able to model complex relationships and represents semantics of add/delete on the relationships.
  - Can handle most situations for modeling using record types and relationship types.
  - Language is navigational; uses constructs like FIND, FIND member, FIND owner,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

Network Model

- **Disadvantages:**
  - Navigational and procedural nature of processing
  - Database contains a complex array of pointers that thread through a set of records.
    - Little scope for automated “query optimization”

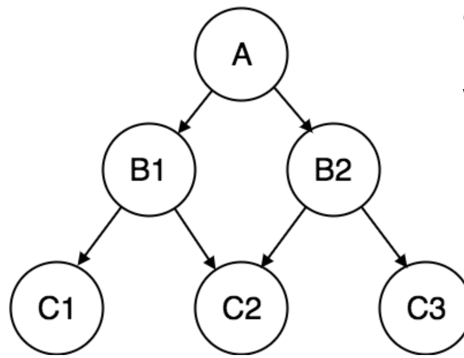
Historical Development of Database Technology

- **Early Database Applications:**
  - The <u>Hierarchical and Network Models were introduced in mid 1960s </u>and dominated during the seventies.
  - A bulk of the worldwide database processing still occurs using these models, particularly, the hierarchical model using IBM's IMS system.



Hierarchical Model

- a top-down structure
- supports one-to-one and
- one-to-many relationships



Network Model

- supports one-to-one
- one-to-many
- and many-to-many relationships
- problem was the inability to support ad hoc queries

Charles William Bachman III (Born on December 11, 1924 – July 13, 2017) was an American computer scientist.

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 (and still has) 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:**
  - Simple to construct and operate
  - Corresponds to a number of natural hierarchically organized domains, e.g., organization ("org") chart
  - Language is simple:
    - Uses constructs like GET, GET UNIQUE, GET NEXT, GET NEXT WITHIN PARENT, etc.

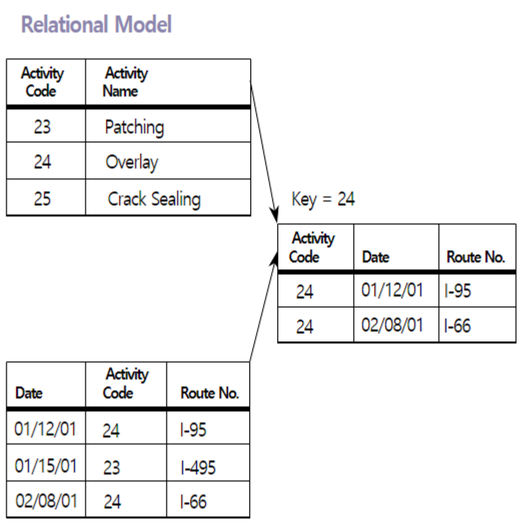
- **Disadvantages:**
  - Navigational and procedural nature of processing
  - Database is visualized as a linear arrangement of records
  - Little scope for "query optimization"

Relational Model

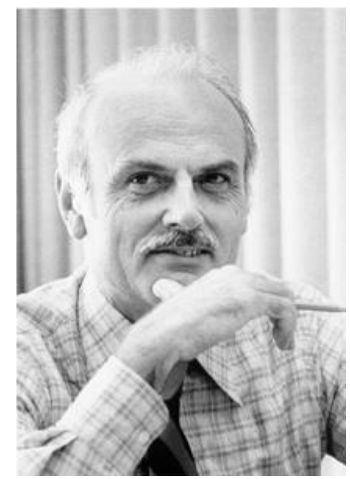
- **Proposed in 1970 by E.F. Codd (IBM), first commercial system in 1981-82.**
- **Now in several commercial products (e.g. DB2, ORACLE, MS SQL Server, SYBASE, INFORMIX).**
- **Several free open source implementations, e.g. MySQL, PostgreSQL**
- **Currently most dominant for developing database applications.**
- **SQL relational standards: SQL-89 (SQL1), SQL-92 (SQL2), SQL-99, SQL3, …**
- **Chapters 5 through 11 describe this model in detail**

Historical Development of Database Technology

- **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.



ORACLE VS IBM



Edgar Frank "Ted" Codd (19 August 1923 – 18 April 2003) was an English computer scientist who, while working for IBM, invented the relational model for database management, the theoretical basis for relational databases and relational database management systems.

Object-oriented Data Models

- Several models have been proposed for implementing in a database system.

- One set comprises models of persistent O-O Programming Languages such as C++ (e.g., in OBJECTSTORE or VERSANT), and Smalltalk (e.g., in GEMSTONE).

- Additionally, systems like 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 12 describes this model.

Object-Relational Models

- The trend to mix object models with relational was started with Informix Universal Server.
- Relational systems incorporated concepts from object databases leading to object-relational.
- Exemplified in the versions of Oracle, DB2, and SQL Server and other DBMSs.
- Current trend by Relational DBMS vendors is to extend relational DBMSs with capability to process XML, Text and other data types.
- The term “Object-relational” is receding in the marketplace.

Chapter Summary

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