

# **Chapter 2**

## **Database System Concepts and Architecture**

# Data Models

- **Data Model:** A set of concepts to describe the *structure* of a database, and certain *constraints* that the database should obey.
- **Data Model Operations:** Operations for specifying database retrievals and updates by referring to the concepts of the data model. Operations on the data model may include *basic operations* and *user-defined operations*.

# 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.
- **Implementation (representational)** data models:  
Provide concepts that fall between the above two, balancing user views with some computer storage details.

# History of Data Models

- Relational Model: proposed in 1970 by E.F. Codd (IBM), first commercial system in 1981-82. Now in several commercial products (DB2, ORACLE, SQL Server, SYBASE, INFORMIX).  
Network Model: the first one to be implemented by Honeywell in 1964-65 (IDS System). Adopted heavily due to the support by CODASYL (CODASYL - DBTG report of 1971). Later implemented in a large variety of systems - IDMS (Cullinet - now CA), DMS 1100 (Unisys), IMAGE (H.P.), VAX -DBMS (Digital Equipment Corp.).
- Hierarchical Data Model: implemented in a joint effort by IBM and North American Rockwell around 1965. Resulted in the IMS family of systems. The most popular model. Other system based on this model: System 2k (SAS inc.)

# History of Data Models

- Object-oriented Data Model(s): 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 O<sub>2</sub>, ORION (at MCC - then ITASCA), IRIS (at H.P.- used in Open OODB).
- Object-Relational Models: Most Recent Trend. Started with Informix Universal Server. Exemplified in the latest versions of Oracle-10i, DB2, and SQL Server etc. systems.

# Hierarchical Model

- **ADVANTAGES:**
  - Hierarchical Model is simple to construct and operate on
  - Corresponds to a number of natural hierarchically organized domains - e.g., assemblies in manufacturing, personnel organization in companies
  - 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"

# 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.

- **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"

# Schemas versus Instances

- **Database Schema:** The *description* of a database. Includes descriptions of the database structure and the constraints that should hold on the database.
- **Schema Diagram:** A diagrammatic 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.
- **Database Instance:** The actual data stored in a database at a *particular moment in time*. Also called **database state** (or **occurrence**).



# 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 when it is loaded
- **Valid State:** A state that satisfies the structure and constraints of the database.
- **Distinction**
  - The **database schema** changes *very infrequently*. The **database state** changes *every time the database is updated*.
  - **Schema** is also called **intension**, whereas **state** is called **extension**.

# Three-Schema Architecture

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

# Three-Schema Architecture

- Defines DBMS schemas at *three levels*:
  - **Internal schema** at the internal level to describe physical storage structures and access paths. 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 level.

# 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 Independence

- **Logical Data Independence:** The capacity to change the conceptual schema without having to change the external schemas and their application programs.
- **Physical Data Independence:** The capacity to change the internal schema without having to change the conceptual schema.

# Data Independence

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). In some DBMSs, separate **storage definition language (SDL)** and **view definition language (VDL)** are used to define internal and external schemas.

# 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 or an Assembly Language.
  - Alternatively, *stand-alone* DML commands can be applied directly (**query language**).



# DBMS Languages

- **High Level or Non-procedural Languages:** e.g., SQL, are *set-oriented* and specify what data to retrieve than how to retrieve. Also called *declarative* languages.
- **Low Level or Procedural Languages:** record-at-a-time; they specify *how* to retrieve data and include constructs such as looping.

# DBMS Interfaces

- Stand-alone query language interfaces.
- Programmer interfaces for embedding DML in programming languages:
  - Pre-compiler Approach
  - Procedure (Subroutine) Call Approach
- User-friendly 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

# 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 accounts, granting authorizations
  - Setting system parameters
  - Changing schemas or access path

# 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.

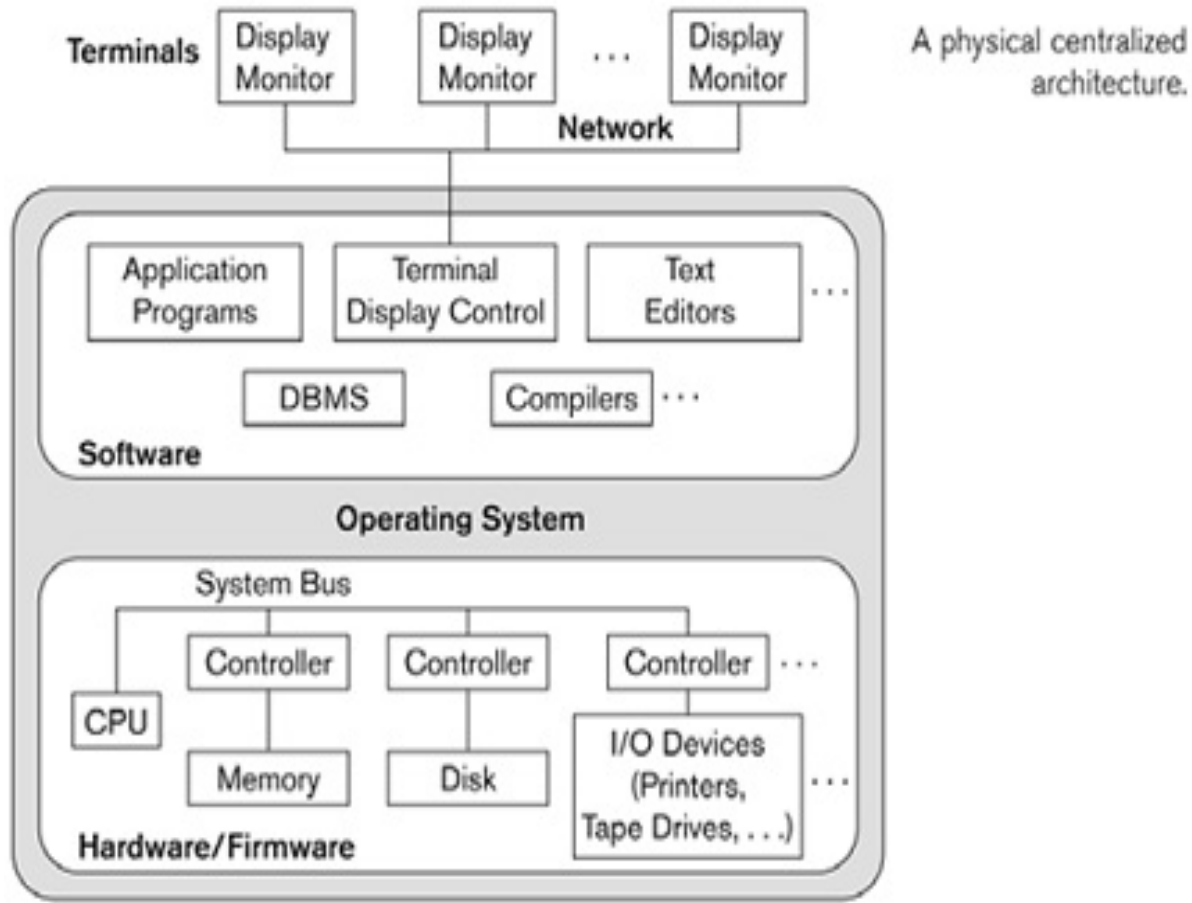
# Other Tools

- **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.
- **Application Development Environments and CASE (computer-aided software engineering) tools:**
  - Examples – Power builder (Sybase), Builder (Borland)

# Centralized and Client-Server Architectures

- **Centralized DBMS:** combines everything into single system including- DBMS software, hardware, application programs and user interface processing software.

# Centralized and Client-Server Architectures



# Basic Client-Server Architectures

- **Specialized Servers with Specialized functions**
- **Clients**
- **DBMS Server**



# Specialized Servers with Specialized functions:

- File Servers
- Printer Servers
- Web Servers
- E-mail Servers

# Clients:

- Provide appropriate interfaces and a client-version of the system to access and utilize the server resources.
- Clients maybe 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
- Sometimes called query and transaction servers

# Two Tier Client-Server Architecture

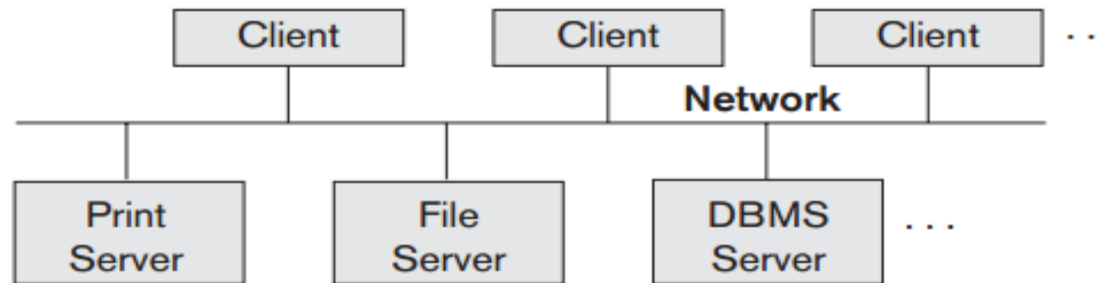
- **User Interface Programs and Application Programs** run on the client side
- Interface called **ODBC (Open Database Connectivity** – Details in Ch 9) provides an Application program interface (API) allow client side programs to call the DBMS. Most DBMS vendors provide ODBC drivers.

# Two Tier Client-Server Architecture

- A client program may connect to several DBMSs.
- Other variations of clients are possible: e.g., in some DBMSs, more functionality is transferred to clients including data dictionary functions, optimization and recovery across multiple servers, etc. In such situations the server may be called the **Data Server**.

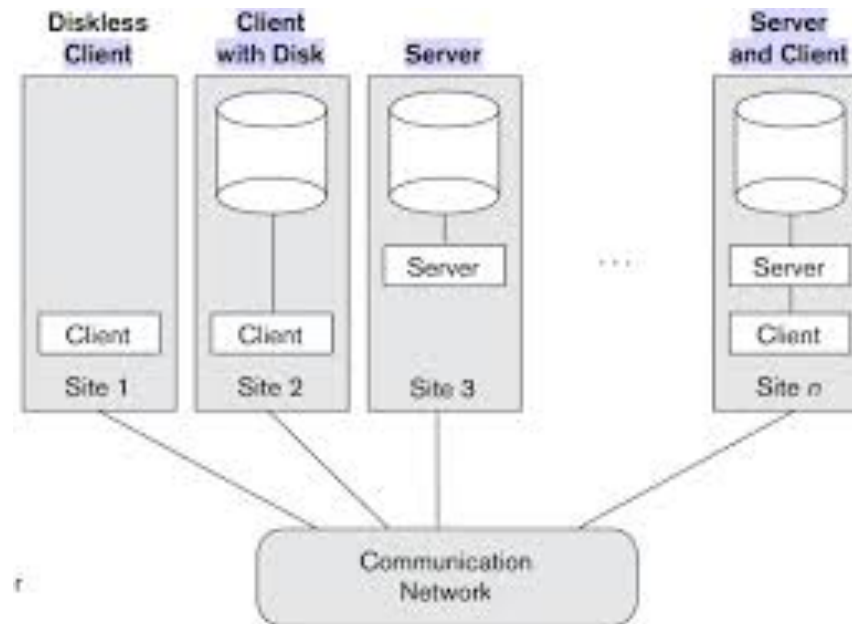
# Two Tier Client-Server Architecture

- Logical two-tier Client-Server Architecture

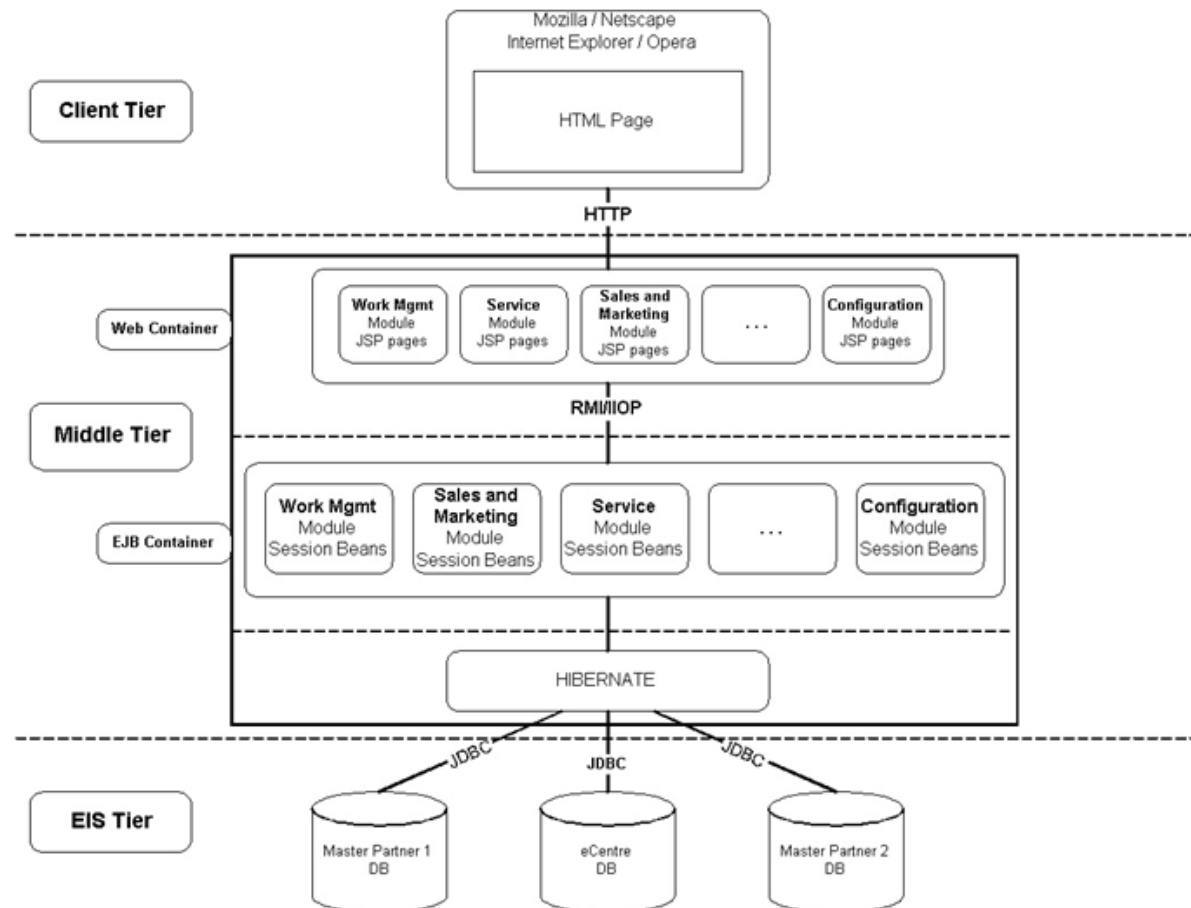


# Two Tier Client-Server Architecture

- Physical two-tier Client-Server Architecture



# Three Tier Client-Server Architecture





# Three Tier Client-Server Architecture

- Common for **Web applications**
- Intermediate Layer called **Application Server** or **Web Server**:
  - stores the web connectivity software and **the rules and business logic (constraints)** part of the application used to access the right amount of data from the database server
  - acts like a conduit for sending partially processed data between the database server and the client.
- **Additional Features- Security**:
  - encrypt the data at the server before transmission
  - decrypt data at the client

# 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 micro-computers) vs. multi-user (most DBMSs).
  - Centralized (uses a single computer with one database) vs. distributed (uses multiple computers, multiple databases)

# Classification of DBMSs

**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.*