Unit I: Introduction to Database

Chapter 01: Databases System Concept 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.

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 model: middle level data model that provides concepts that fall between the above two, balancing user views with some computer storage details.

Example of Data Model

• Relational Model: proposed in 1970 by E.F. Codd (IBM), first commercial system in 1981-82. Now in several commercial products (ORACLE, SYBASE, INFORMIX, CA-INGRES).

• 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).

• 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.)

• 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 O2, ORION (at MCC - then ITASCA), IRIS (at H.P.- used in Open OODB).

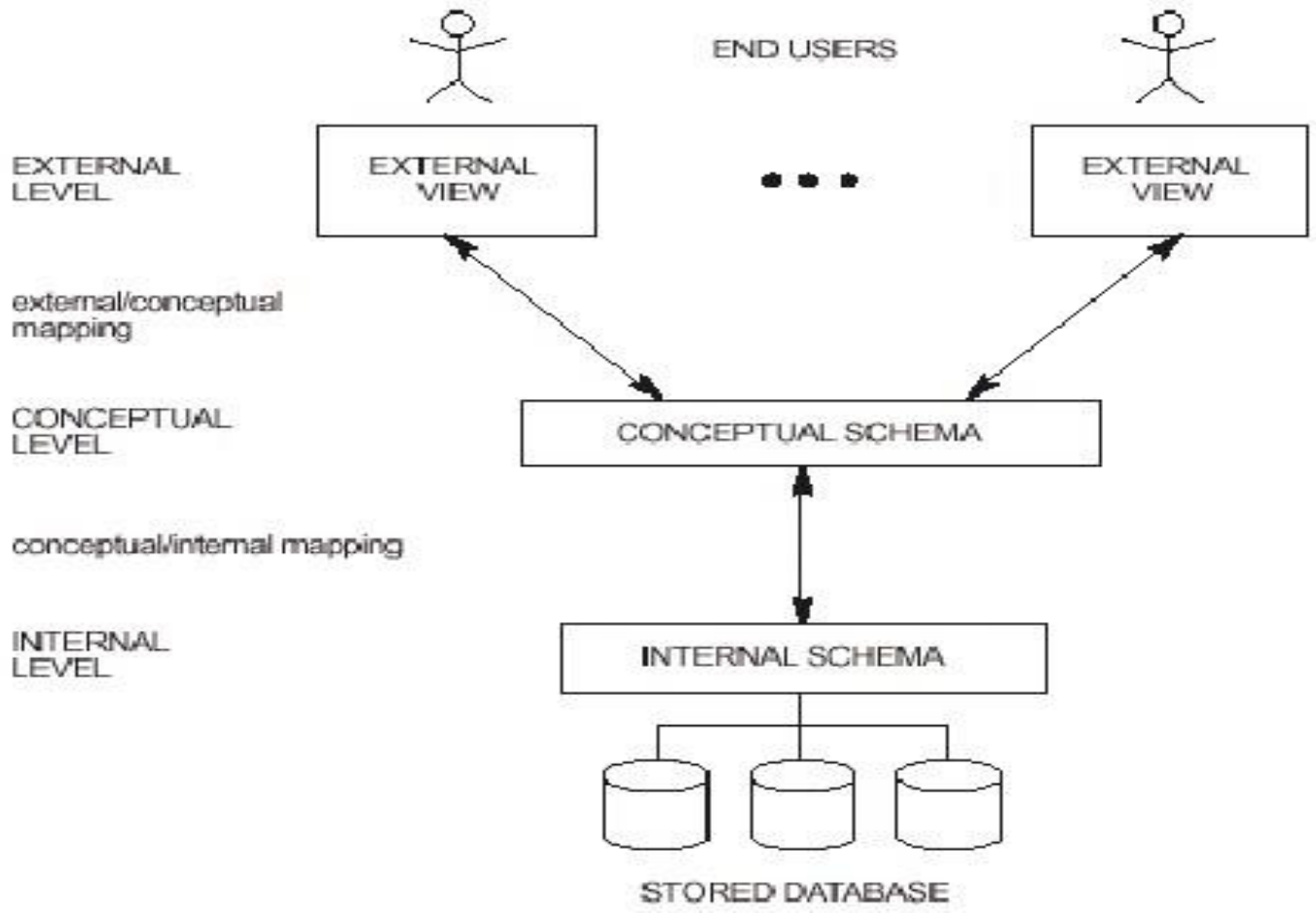
- Object-Relational Models : Most Recent Trend. Exemplified in ILLUSTRA and UNiSQL systems.

Schemas versus Instances

- **Database Schema**: is the *description* of a database.
- **Schema Diagram**: is used to display some aspects of a database schema such as the names of record types and data items, and some constraints that hold on the database.
- **Schema Construct**: is a component of the schema or an object within the schema, e.g., STUDENT, COURSE.
- **Database Instance**: is the actual data stored in a database at the particular instant of time. Also called **database state**.

Every time we update the database (insert/delete a record or change the of data item in a record), we change one state of the database to another state.

Three Schema Architecture



Three-Schema Architecture

To achieve and visualize DBMS characteristics such as Data Abstraction, and multiple views of data, the three-schema architecture was designed for database systems.

Three-Schema defines DBMS schemas at *three levels*:

- ✓ **External schemas** at the external level to describe the various user views. Usually uses the same data model as the conceptual level.

- ✓ **Conceptual schema** at the conceptual level to describe the structure and constraints for the database for all users. Uses a *conceptual* or an *implementation* data model.
- ✓ **Internal schema** at the internal level to describe physical storage structures and access paths. Typically uses a *physical* data model.

Mappings among schema levels are needed to transform requests and data. Application programs refer to an external schema, and are mapped by the DBMS to the internal schema for execution.

Data Independence

What/how data independence are achieve by three-Schema Architecture?

- ✓ **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.
- ✓ 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

- **For DBMS with NO clear separation between levels:**
  - **DDL** is used to specify both the *conceptual & internal schemas* and their mappings.
- **For DBMS with true 3-schema Architecture (clear separation between levels):**
  - **Data Definition Language** (**DDL**): used to specify the *conceptual schema* of a database.
  - **Storage Definition Language** (**SDL**) : used to specify the internal schema of a database

    Either DDL or SDL is used to specify for mapping from conceptual to the internal
  - **View definition Language** (**VDL**) are used to specify the external schema of a database and their mapping to the conceptual schema.
- **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++/ Java/etc.
  - Alternatively, *stand-alone* DML commands can be applied directly (**query language**).

- Each DBMS languages will have its **own compiler** to process their statements & identify the description of the schema constructs and to store the schema description in the DBMS

Comprehensive Integrated DBMS Languages(SQL)

- **Structure Query Language(SQL):** is comprehensive integrated Database Language (mainly for Relational DBMS) which represents a combination of DDL, VDL and DML.
  - ✓ It excludes SDL to keep it at the conceptual & external levels only.
  - ✓ SDL is kept as separate for defining physical storage structures to fine-tune the performance of the database system.
- **SQL can be referred as:**
  - ➢ **High Level** or **Non-procedural Language** since it is *set-oriented* and specify what data to retrieve than how to retrieve. Also called *declarative* languages.
  - ➢ **Low Level** or **Procedural Languages** since it is embedded in a general purpose programming langauge (c/c++/java/etc..) to specify *how* to retrieve data and include constructs such as looping.
- SQL is the **most popular** Database language for RDBMS as RDBMS is **widely used** in most of the current **database applications**
- Therefore, in the current DBMS, DDL, SDL, VDL and DML are usually **not** considered as **distinct languages**.

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

Other DBMS Interfaces

- 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

Centralized and Client-Server Architectures

- **Centralized DBMS:** everything perform by single system including- DBMS software, hardware, application programs and user interface program.
- **Basic Client-Server Architectures**
  - Specialized Servers with Specialized functions
    - ✓ File Servers
    - ✓ Printer Servers

- ✓ Web Servers
- ✓ E-mail Servers

- Clients
- DBMS Server

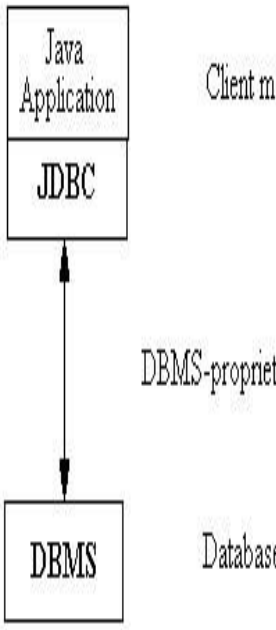
Client-Server Architectures

- **Clients:**
  - Provide appropriate interfaces 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
- use Application program interface (API) that allow client side programs to call the DBMS. e.g of API:
  - ✓ **ODBC (Open Database Connectivity) ;**
  - ✓ **JDBC**
- A client program may connect to several DBMSs.



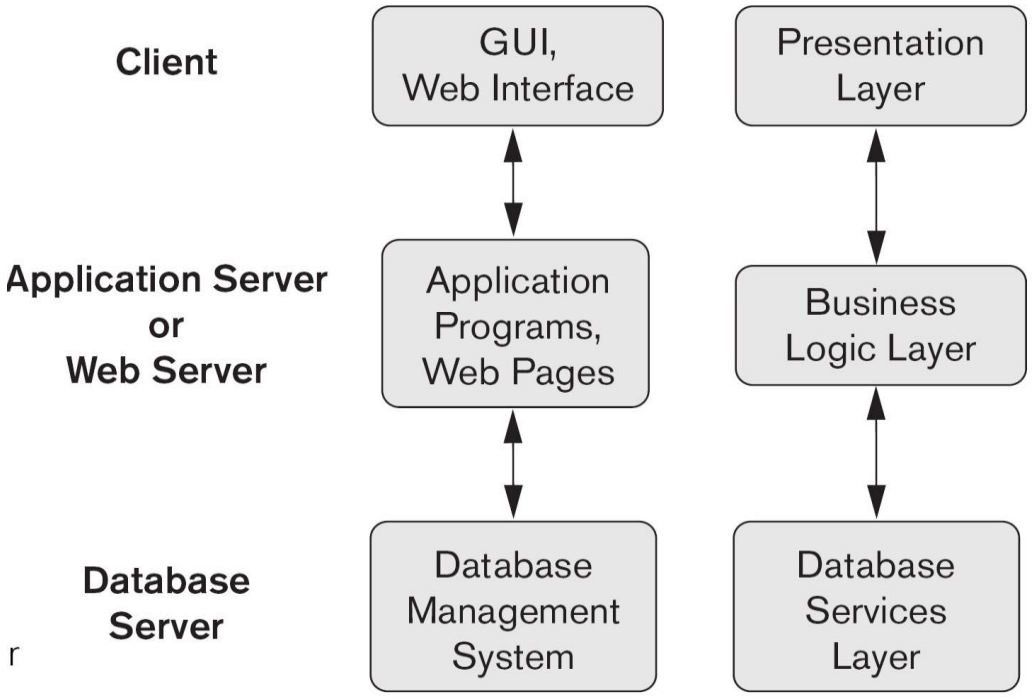
Three Tier Client-Server Architecture:

Common for **Web applications**
Intermediate Layer called **Application Server** or **Web Server:**

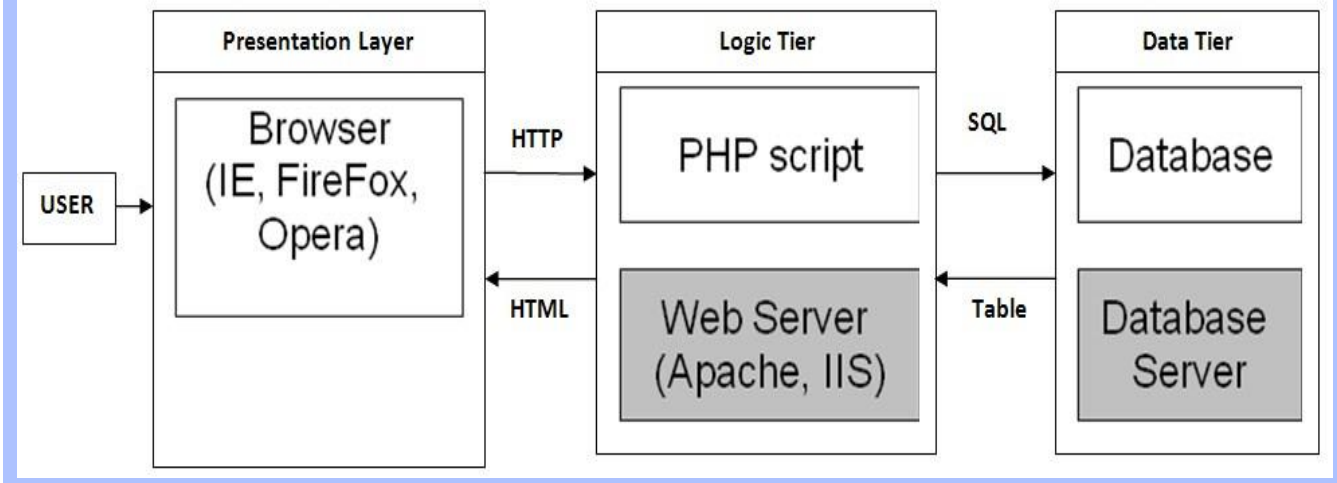
- stores the web connectivity software and **the rules and business logic (constraints)** part of the application used to access the right amount of data from the database server
- acts like a conduit for sending partially processed data between the database server and the client.

Additional Features- Security:

- encrypt the data at the server before transmission
- decrypt data at the client


Client
GUI,
Web Interface
Presentation
Layer
Application Server
or
Web Server
Application
Programs,
Web Pages
Business
Logic Layer
Database
Server
Database
Management
System
Database
Services
Layer

r


Presentation Layer
Logic Tier
Data Tier
USER
Browser
(IE, FireFox,
Opera)
HTTP
HTML
PHP script
Web Server
(Apache, IIS)
SQL
Table
Database
Database
Server