# Lecture 2
## Database System Concepts and Architecture

# Agenda

- Data Models
- Categories of Data Models
- Schema
- Database state
- Three-Schema Architecture
- DBMS Languages
- Client/Server Architectures for DBMSs

# Data Models

# Data Models

- One fundamental characteristic of the database approach is that it provides some level of data abstraction.

- **Data abstraction** generally refers to:
  - **Suppression of details** of data organization and storage.
  - **Highlighting of the essential features** for an improved understanding of data.

# Data Models

- ➢ **Data abstraction is achieved by a Data model.**

- ➢ What is a Data Model?
  - A collection of **concepts** that can be used to describe the *structure of a database.*
  - By *structure of a database* we mean the **data types**, **relationships**, and **constraints** that apply to the data.
  - Most data models also include a set of **basic operations** for specifying *retrievals* and *modifications* on the database.

# Categories of Data Models

# Categories of Data Models

Conceptual data models
(High Level)

Representational data models
(Implementational)

Physical data models
(Low Level)

# Categories of Data Models

| Conceptual data models (High Level) | Representational data models (Implementational) | Physical data models (Low Level) |
|---|---|---|
| Provide concepts that are close to the way many users perceive data | Provide concepts that **may** be easily understood by end users and **aren't too far** from how data is organized in computer storage | Provide concepts that describe the details of how data is stored on the computer storage media |

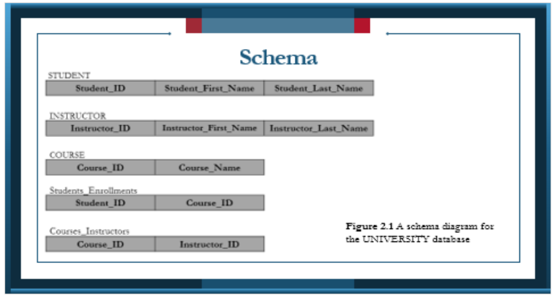
# Categories of Data Models

- Conceptual data models → **Entity–Relationship model**
- Representational data models → **Relational data model**
- Physical data models → **Access path**

# Schema

# Schema

- In a data model, it is important to distinguish between the *description* of the database and the *database itself.*
- The description of a database is called the **database schema**, which is specified during database design and is not expected to change frequently.
- A schema diagram for the UNIVERSITY database given before:



# Schema

STUDENT

| | | |
|---|---|---|
| Student_ID | Student_First_Name | Student_Last_Name |

INSTRUCTOR

| | | |
|---|---|---|
| Instructor_ID | Instructor_First_Name | Instructor_Last_Name |

COURSE

| | |
|---|---|
| Course_ID | Course_Name |

Students_Enrollments

| | |
|---|---|
| Student_ID | Course_ID |

Courses_Instructors

| | |
|---|---|
| Course_ID | Instructor_ID |

**Figure 2.1** A schema diagram for the UNIVERSITY database

# Schema

- The schema diagram displays the **structure of each record** but not the **actual records**.
- A schema diagram displays only *some aspects* of a schema, such as the names of record types and data items, and *some types of constraints*.
- Other aspects are not specified in the schema diagram; for example, the **data type** of each data item isn't present.

# Database state

# Database state

- The actual data in a database may change frequently.
- For example, the UNIVERSITY database changes every time we add a new student or add a new course.
- The data in the database at a particular moment in time is called a **database state** or **snapshot**.

# Database state

- Every time we insert or delete a record or change the value of a data item in a record, we **change one state of the database into another state**.

- When we **define** a new database, we specify its database schema only to the DBMS.

- At this point, the corresponding database state is the *empty state* with no data.

# Database state

- We get the *initial state* of the database when the database is first populated or loaded with the initial data.
- From then on, every time a modification operation is applied to the database, we get another database state.
- At any point in time, the database has a *current state*.

# Database state

- The DBMS is partly responsible for ensuring that every state of the database is a **valid state**—that is, a state that satisfies the structure and constraints specified in the schema.

- The DBMS stores the descriptions of the schema constructs and constraints—also called the **meta-data**—in the DBMS catalog so that DBMS software can refer to the schema whenever it needs to.

# Three-Schema Architecture

# Three-Schema Architecture

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

- In this architecture, schemas can be defined at the following three levels:
  - **Internal Level**
  - **Conceptual Level**
  - **External** or **View Level**

# Three-Schema Architecture

- **Internal Level**
  - Has an **internal schema**, which describes the physical storage structure of the database.
  - The internal schema uses a **physical data model** and describes the complete details of data storage and access paths for the database.

# Three-Schema Architecture

## ➢ Conceptual Level

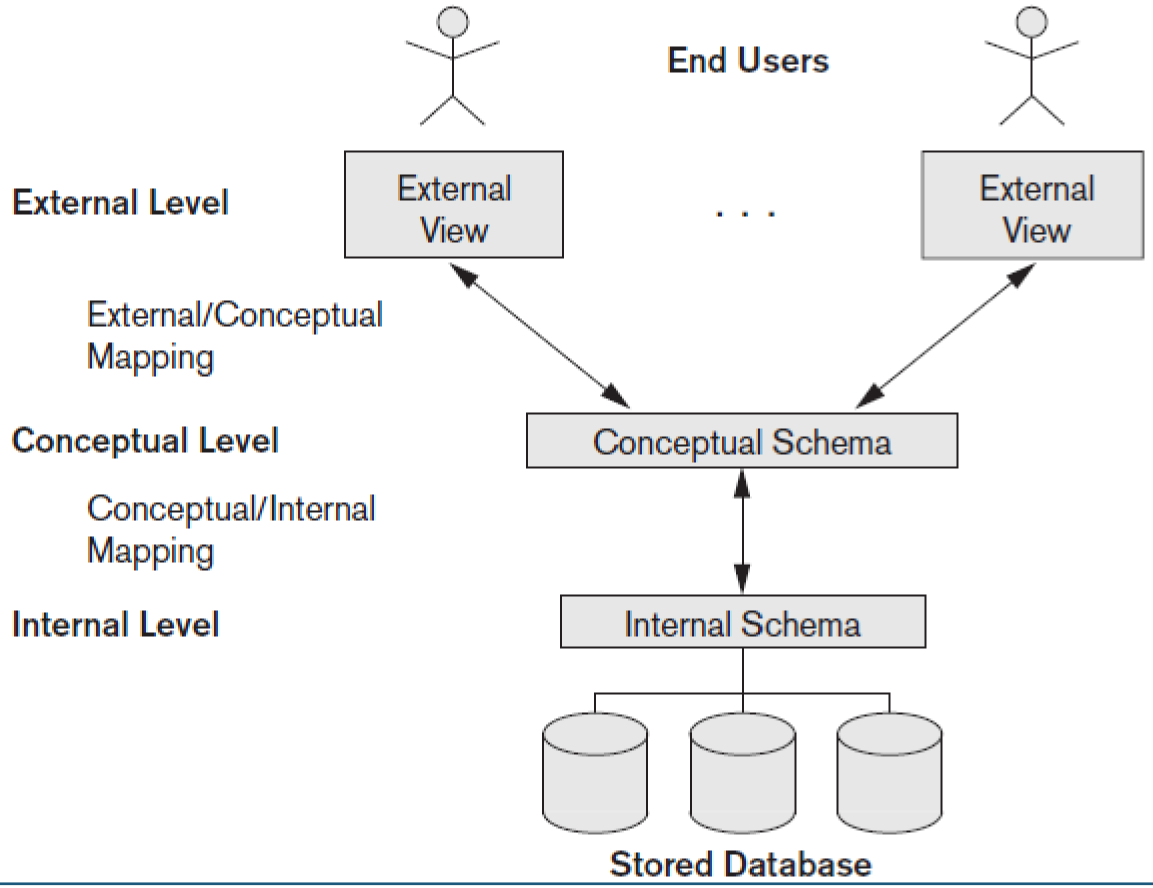
- Has a **conceptual schema**, which describes the structure of the whole database for users.
- The conceptual schema hides the details of physical storage structures and concentrates on describing entities, data types, relationships, user operations, and constraints.
- Usually, a **representational data model** is used to describe the conceptual schema.

# Three-Schema Architecture

- **External** or **View Level**
  - Includes a number of **external schemas** or **user views**.
  - 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.
  - Each external schema is typically uses a **representational data model**.

# Three-Schema Architecture

**Figure 2.2**
The three-schema architecture.



# Three-Schema Architecture

- For the UNIVERSITY database given, show an example of a **Conceptual Schema** and an **External Schema**.

# DBMS Languages

# DBMS Languages

- In many DBMSs where no strict separation of levels is maintained, one language, called the **data definition language (DDL)**, is used to define conceptual and internal schemas.

- In DBMSs where a clear separation is maintained between the conceptual and internal levels, the DDL is used to specify the conceptual schema only. Another language, the **storage definition language (SDL)**, is used to specify the internal schema.

# DBMS Languages

- In most relational DBMSs today, there *is no specific language* that performs the role of SDL. Instead, the internal schema is specified by a combination of functions, parameters, and specifications related to storage of files.

- For a true three-schema architecture, we would need a third language, the **view definition language (VDL)** to specify user views and their mappings to the conceptual schema.

# DBMS Languages

- In most DBMSs *the DDL is used to define both conceptual and external schemas.*

- In relational DBMSs, **Structured Query Language (SQL)** is used in the role of VDL to define user or application views.

- Once the database schemas are compiled and the database is populated with data, users must have some means to manipulate the database (i.e. *retrieval*, *insertion*, *deletion*, and *updating* of the data). DBMS provides a language called the **data manipulation language (DML)** for these purposes.

# DBMS Languages

- **DDL:** Conceptual schemas
  Internal schemas (if no separation)
  External schemas
- **SDL:** Internal schemas
- **VDL:** User views and their mappings
- **SQL:** User or application views
- **DML:** Manipulate the database

# DBMS Languages

- In current DBMSs, the preceding types of languages are usually *not considered distinct languages*; rather, a comprehensive integrated language is used that includes constructs for conceptual schema definition, view definition, and data manipulation. Storage definition is typically kept separate.
- A typical example of a comprehensive database language is the **SQL** relational database language, which represents a combination of *DDL*, *VDL*, and *DML*, as well as statements for constraint specification, schema evolution, and many other features.

# Client/Server Architectures for DBMSs

# Client/Server Architectures for DBMSs

- DBMS was a **centralized DBMS** in which all the DBMS functionality, application program execution, and user interface processing were carried out on one machine.

- Gradually, DBMS systems started to exploit the available processing power at the user side, which led to **client/server DBMS** architectures.

# Client/Server Architectures for DBMSs

- The idea is to define **specialized servers** with specific functionalities.
- For example, we can have:
  - **File server** that maintains the files.
  - **Printer server** connected to various printers; all print requests by the clients are forwarded to this machine.
  - **Web servers** or **e-mail servers.**
- The **client machines** provide the user with the appropriate interfaces to utilize these servers as well as with local processing power to run local applications.

# Client/Server Architectures for DBMSs

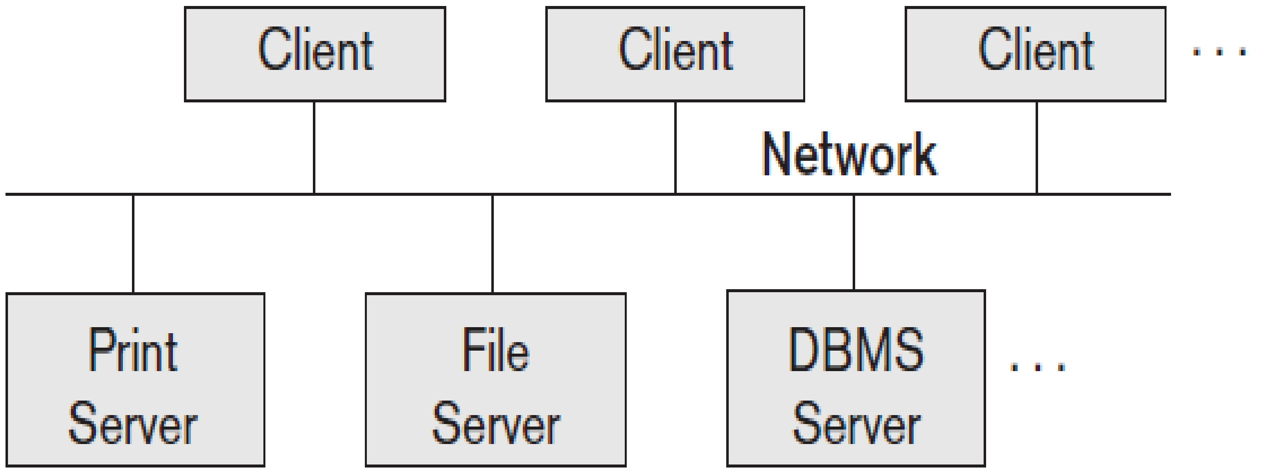



**Figure 2.3** A Client/Server Architecture

# Thank You