Database System Concepts and Architecture

Dr. Odelu Vanga

Indian Institute of Information Technology Sri City

http://www.iiits.ac.in/people/regular-faculty/dr-odelu-vanga/

Outline

- Traditional File Processing and Drawbacks
- Basic Concepts of Database System
- Database System Architecture

Traditional File Processing
(University Example)

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

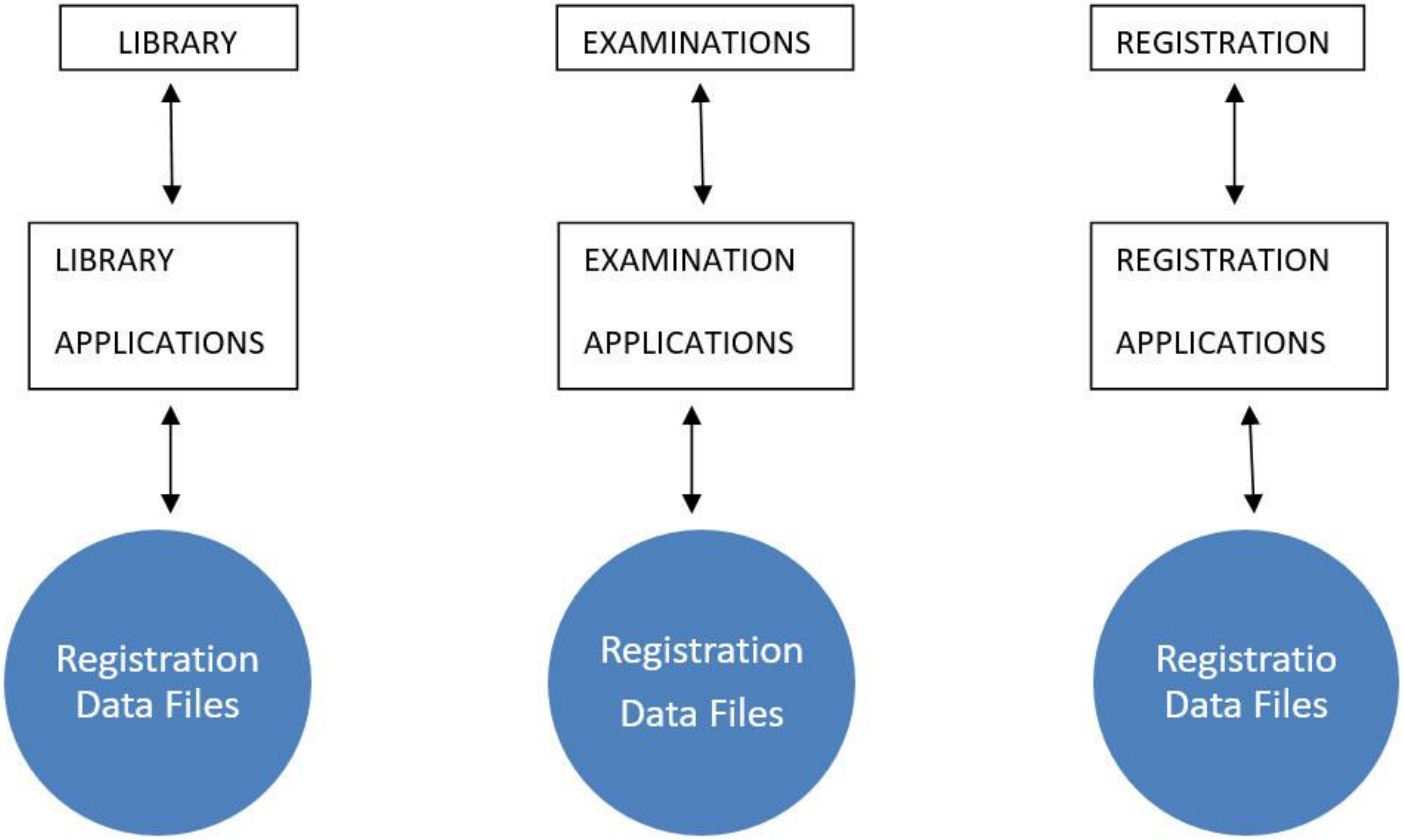
GRADE_REPORT

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

Traditional File Processing



Data Manipulation

Queries:

- Retrieve transcript - a list of all courses and grades - of `Smith'
- List the names of students who took the section of the `Database' course offered in fall 2008 and their grades in that section
- List the prerequisites of the `Database' course

Updates:

- Change the class of `Smith' to sophomore
- Create a new section for the `Database' course for this semester
- Enter a grade of `A' for `Smith' in the `Database' section of last semester

Drawbacks of Traditional File System

- Data Redundancy and Inconsistency
- Difficulty in Accessing Data
- Data Isolation
- Integrity Problems
- Atomicity Problems
- Concurrent-Access Anomalies
- Security Problems

Data Redundancy and Inconsistency

- **Different programmers**
  - create files
  - application programs.
  - different file structures.
  - several programming languages.
- **Information duplicated in several places**, called **redundancy**. It leads to higher storage and access cost.
- **In addition, it may lead to data inconsistency**.

Difficulty in Accessing Data

- University clerks needs to find out the names of all students who live within a particular postal-code area.
- The clerk asks the data-processing department to generate such a list.
- The university clerk has now two choices:
  - obtain the list of all students and extract the needed information manually
  - ask a programmer to write the necessary application program
- Both alternatives are obviously unsatisfactory.
- The point here is that conventional file-processing environments do not allow needed data to be retrieved in a convenient and efficient manner.

Data Isolation

- Data are scattered in various files
- Files may be in different formats
- So, writing new application programs to retrieve the appropriate data is difficult

Integrity Problems

- The data values stored in the database must satisfy certain types of **consistency constraints**.
- Suppose the university maintains an account for each department, and records the balance amount in each account.
- Suppose also that the university requires that
  - the account balance of a department may never fall below zero.
- Developers enforce these constraints in the system by adding appropriate code in the various application programs.
- However, when new constraints are added, it is difficult to change the programs to enforce them.
- The problem is compounded when constraints involve several data items from different files.

Atomicity Problems

- A computer system, like any other device, is subject to failure.
- In many applications, it is crucial that, if a failure occurs, the data be restored to the consistent state that existed prior to the failure.
- **Transfer $500** from the department *A* to department *B's* account.
- If a system failure occurs during the execution of the program
  - It may possible that $500 debited from department *A'* account, but not credited to department *B'* account.
  - It resulting in an **inconsistent database state**.
- Clearly, it is essential to database consistency that either both the credit and debit occur, or that neither occur.
- That is, the funds transfer must be ***atomic*** - it must happen in its entirety or not at all.
- **It is difficult to ensure atomicity in a conventional file-processing system.**

Concurrent-Access Anomalies

- For the sake of overall performance of the system and faster response, many systems allow multiple users to update the data simultaneously.
- Consider department *A*, with an account balance of $10,000.
- If two department clerks debit the department *A*'s account by, say $500 and $100, respectively, at almost exactly the same time.
- If the two programs run concurrently, they may both read the value $10,000, and write back $9500 and $9900, respectively.
- Depending on which one writes the value last, the account balance of department *A* may contain either $9500 or $9900, rather than the correct value of $9400.
- But, supervision is difficult to provide because data may be accessed by many different application programs that have not been coordinated previously.

Security Problems

- Not every user of the database system should be able to access all the data.
- For example, in a university, payroll personnel need to see only that part of the database that has financial information.
- They do not need access to information about academic records.
- But, since application programs are added to the file-processing system in an ad hoc manner, enforcing such security constraints is difficult.

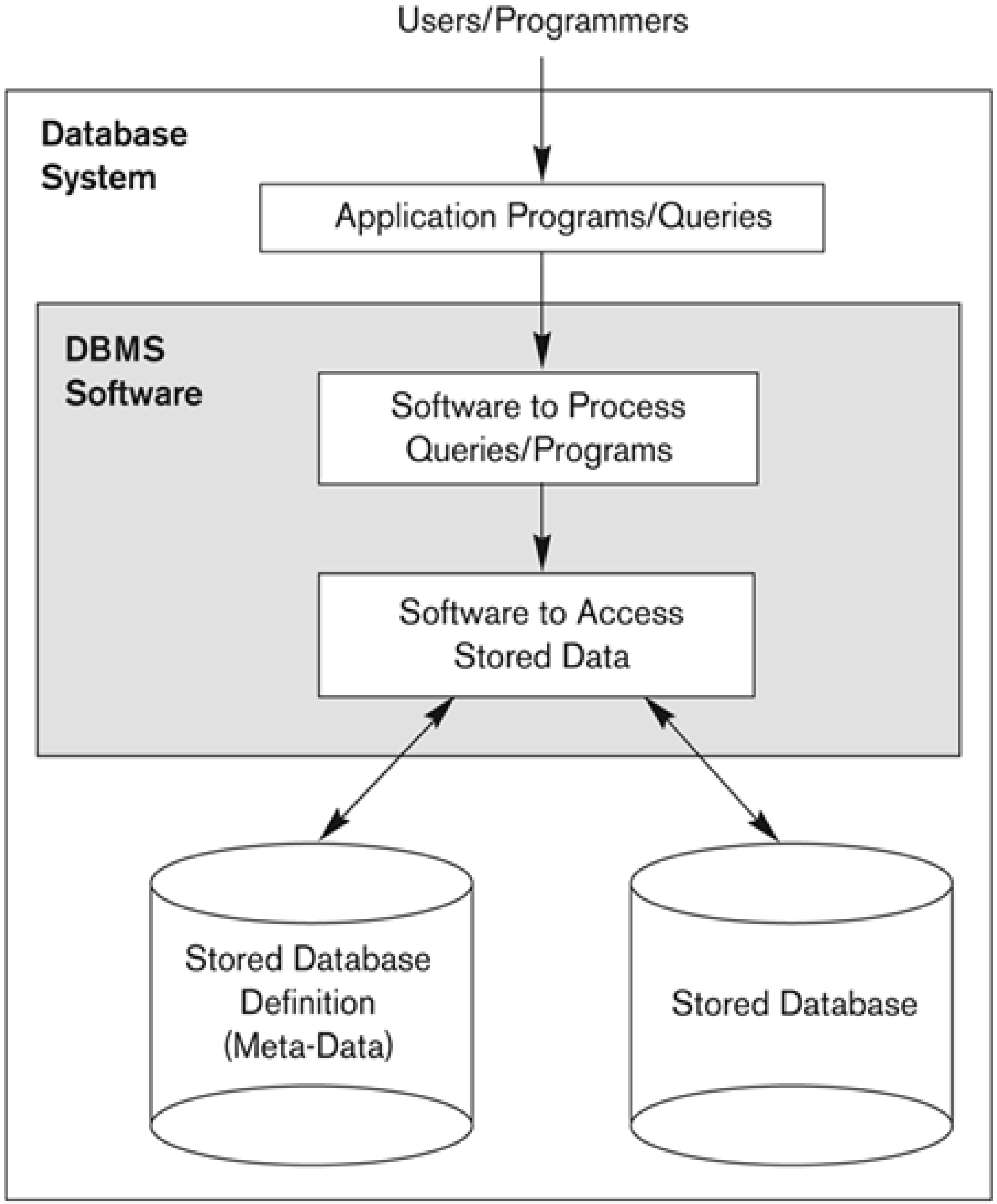
Types of Databases and Applications

- **Traditional Applications:**
  - Numeric and Textual Databases
- **More Recent Applications:**
  - Multimedia Databases
  - Geographic Information Systems (GIS)
  - Data Warehouses
  - Real-time and Active Databases
  - Many other applications

Basic Definitions

- **Database:**
  - A collection of related data.
- **Data:**
  - Known facts that can be recorded and have an implicit meaning.
- **Mini-world:**
  - Some part of real-world about which data is stored in a database.
  - For example, student grades and transcripts at a university.
- **Database Management System (DBMS):**
  - A software package/system to facilitate the creation and maintenance of a computerized database.
- **Database System:**
  - The DBMS software together with the data itself.
  - Sometimes, the applications are also included.

Simplified database system environment



The database definition or descriptive information is stored by the DBMS in the form of a database catalog or dictionary;

it is called **meta-data**

Typical DBMS Functionality

- ***Define*** a particular database in terms of its data types, structures, and constraints
- ***Construct*** or **Load** the initial database contents on a secondary storage medium
- ***Manipulating*** the database:
  - **Retrieval:** Querying, generating reports
  - **Modification:** Insertions, deletions and updates to its content
  - **Accessing** the database through Web applications
- ***Processing*** and ***Sharing*** by a set of concurrent users and application programs -- yet, keeping all data valid and consistent

Typical DBMS Functionality

- **Other features:**
  - Protection or Security measures to prevent unauthorized access
  - "Active" processing to take internal actions on data
  - Presentation and Visualization of data
  - Maintaining the database and associated programs over the lifetime of the database application
    - Called database, software, and system maintenance

Example of a Database (with a Conceptual Data Model)

- **Mini-world for the example:**
  - Part of a UNIVERSITY environment.
- **Some mini-world *entities*:**
  - STUDENTs
  - COURSEs
  - SECTIONs (of COURSEs)
  - (academic) DEPARTMENTs
  - INSTRUCTORs

Example of a Database (with a Conceptual Data Model)

- **Some mini-world *relationships*:**
  - SECTIONs *are of specific* COURSEs
  - STUDENTs *take* SECTIONs
  - COURSEs *have prerequisite* COURSEs
  - INSTRUCTORs *teach* SECTIONs
  - COURSEs *are offered by* DEPARTMENTs
  - STUDENTs *major in* DEPARTMENTs

- Note: The above entities and relationships are typically expressed in a conceptual data model, such as the ENTITY-RELATIONSHIP data model.

Example of a simple database

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

Main Characteristics of the Database Approach

- **Self-describing nature of a database system:**
  - A DBMS **catalog** stores the description of a particular database (e.g. data structures, types, and constraints)
  - The description is called **meta-data**.
  - This allows the DBMS software to work with different database applications.
- **Insulation between programs and data:**
  - Called **program-data independence**.
  - Allows changing data structures and storage organization without having to change the DBMS access programs.

Example of a simplified database catalog

RELATIONS

Relation_name	No_of_columns
STUDENT	4
COURSE	4
SECTION	5
GRADE_REPORT	3
PREREQUISITE	2

COLUMNS

Column_name	Data_type	Belongs_to_relation
Name	Character (30)	STUDENT
Student_number	Character (4)	STUDENT
Class	Integer (1)	STUDENT
Major	Major_type	STUDENT
Course_name	Character (10)	COURSE
Course_number	XXXXNNNN	COURSE
....	....	.....
....	....	.....
....	....	.....
Prerequisite_number	XXXXNNNN	PREREQUISITE

Main Characteristics of the Database Approach (continued)

- **Data Abstraction:**
  - A **data model** is used to hide storage details and present the users with a conceptual view of the database.
  - Programs refer to the data model constructs rather than data storage details
- **Support of multiple views of the data:**
  - Each user may see a different view of the database, which describes **only** the data of interest to that user.

Main Characteristics of the Database Approach (continued)

- **Sharing of data and multi-user transaction processing:**
  - Allowing a set of **concurrent users** to retrieve from and to update the database.
  - *Concurrency control* within the DBMS guarantees that each **transaction** is correctly executed or aborted
  - *Recovery* subsystem ensures each completed transaction has its effect permanently recorded in the database
  - **OLTP** (Online Transaction Processing) is a major part of database applications. This allows hundreds of concurrent transactions to execute per second.

Database Users

- **Users may be divided into**
  - actually **use and control** the database content
  - Other- **design, develop and maintain** database applications (called "Actors on the Scene")
  - who design and develop the DBMS software and related tools, and the computer systems operators (called "Workers Behind the Scene").

Database Users

- **Actors on the scene**
  - **Database administrators:**
    - Responsible for authorizing access to the database, for coordinating and monitoring its use, acquiring software and hardware resources, controlling its use and monitoring efficiency of operations.
  - **Database Designers:**
    - Responsible to define the content, the structure, the constraints, and functions or transactions against the database.
    - They must communicate with the end-users and understand their needs.

Categories of End-users

- Actors on the scene (continued)
  - **End-users:** They use the data for queries, reports and some of them update the database content. End-users can be categorized into:
    - **Casual**: access database occasionally when needed
    - **Naïve** or **Parametric**: they make up a large section of the end-user population.
      - They use previously well-defined functions in the form of "canned transactions" against the database.
      - Examples are bank-tellers or reservation clerks who do this activity for an entire shift of operations.

Categories of End-users (continued)

- **Sophisticated:**
  - These include business analysts, scientists, engineers, others thoroughly familiar with the system capabilities.
  - Many use tools in the form of software packages that work closely with the stored database.
- **Stand-alone:**
  - Mostly maintain personal databases using ready-to-use packaged applications.
  - An example is a tax program user that creates its own internal database.
  - Another example is a user that maintains an address book

Advantages of Using the Database Approach

- Controlling redundancy in data storage and in development and maintenance efforts.
  - Sharing of data among multiple users.
- Restricting unauthorized access to data.
- Providing persistent storage for program Objects
  - In Object-oriented DBMSs
- Providing Storage Structures (e.g. indexes) for efficient Query Processing

Advantages of Using the Database Approach (continued)

- Providing backup and recovery services.
- Providing multiple interfaces to different classes of users.
- Representing complex relationships among data.
- Enforcing integrity constraints on the database.
- Drawing inferences and actions from the stored data using deductive and active rules

Additional Implications of Using the Database Approach

- Potential for enforcing standards:
  - This is very crucial for the success of database applications in large organizations.
  - **Standards** refer to
    - data item names
    - display formats
    - report structures
    - meta-data (description of data)
    - Web page layouts, etc.
- Reduced application development time:
  - Incremental time to add each new application is reduced.

Additional Implications of Using the Database Approach (continued)

- Flexibility to change data structures:
  - Database structure may evolve as new requirements are defined.
- Availability of current information:
  - Extremely important for on-line transaction systems such as airline, hotel, car reservations.
- Economies of scale:
  - Wasteful overlap of resources and personnel can be avoided by consolidating data and applications across departments.

Historical Development of Database Technology

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

Historical Development of Database Technology (continued)

- Object-oriented and emerging applications:
  - Object-Oriented Database Management Systems (OODBMSs) were introduced in late 1980s and early 1990s to cater to the need of complex data processing in CAD and other applications.
    - Their use has not taken off much.
  - Many relational DBMSs have incorporated object database concepts, leading to a new category called *object-relationa*l DBMSs (ORDBMSs)
  - *Extended relational* systems add further capabilities (e.g. for multimedia data, XML, and other data types)

Extending Database Capabilities

- New functionality is being added to DBMSs in the following areas:
  - Scientific Applications
  - XML (eXtensible Markup Language)
  - Image Storage and Management
  - Audio and Video Data Management
  - Data Warehousing and Data Mining
  - Spatial Data Management
  - Time Series and Historical Data Management

- The above gives rise to *new research and development* in incorporating new data types, complex data structures, new operations and storage and indexing schemes in database systems.

When not to use a DBMS

- **Main inhibitors (costs) of using a DBMS:**
  - High initial investment and possible need for additional hardware.
  - Overhead for providing generality, security, concurrency control, recovery, and integrity functions.
- **When a DBMS may be unnecessary:**
  - If the database and applications are simple, well defined, and not expected to change.
  - If access to data by multiple users is not required.
- **When no DBMS may suffice:**
  - If the database system is not able to handle the complexity of data because of modeling limitations.
  - If database users need special operations not supported by DBMS.

Summary

- Traditional File Processing Their Drawbacks
- Types of Databases and Database Applications
- Basic Definitions
- Typical DBMS Functionality
- Example of a Database (UNIVERSITY)
- Main Characteristics of the Database Approach
- Database Users
- Advantages of Using the Database Approach
- When Not to Use Databases

BREAK

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

Data Models

- **Data Model:**
  - A set of concepts to describe
    - ***structure*** of a database
    - *operations* for manipulating these structures
    - *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)
    - ***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.

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

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

Schemas versus Instances

- **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 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

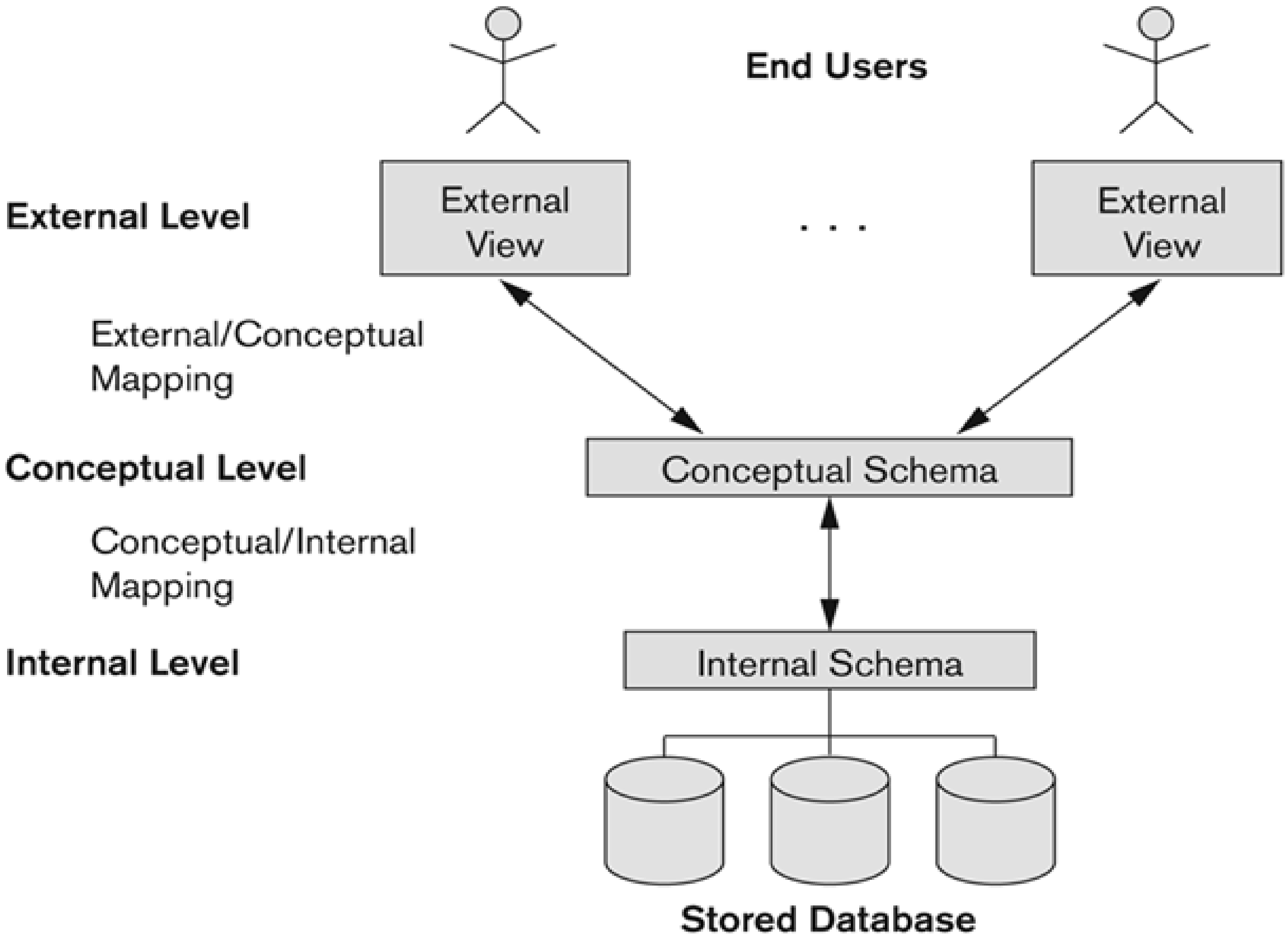
Three-Schema Architecture

- 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.

The three-schema architecture



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 extracted from the internal DBMS level is reformatted to match the user's external view

For example:

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

Data Independence (continued)

- 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)
- Data Manipulation Language (DML)
  - High-Level or Non-procedural Languages: These include the relational language SQL
  - Low Level or Procedural Languages:
    - These must be embedded in a programming language

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.
    - SDL is typically realized via DBMS commands provided to the DBA and database designers

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, 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:**
  - 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.

DBMS Programming Language Interfaces

- Programmer interfaces for embedding DML in a programming languages:
  - **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

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 access paths

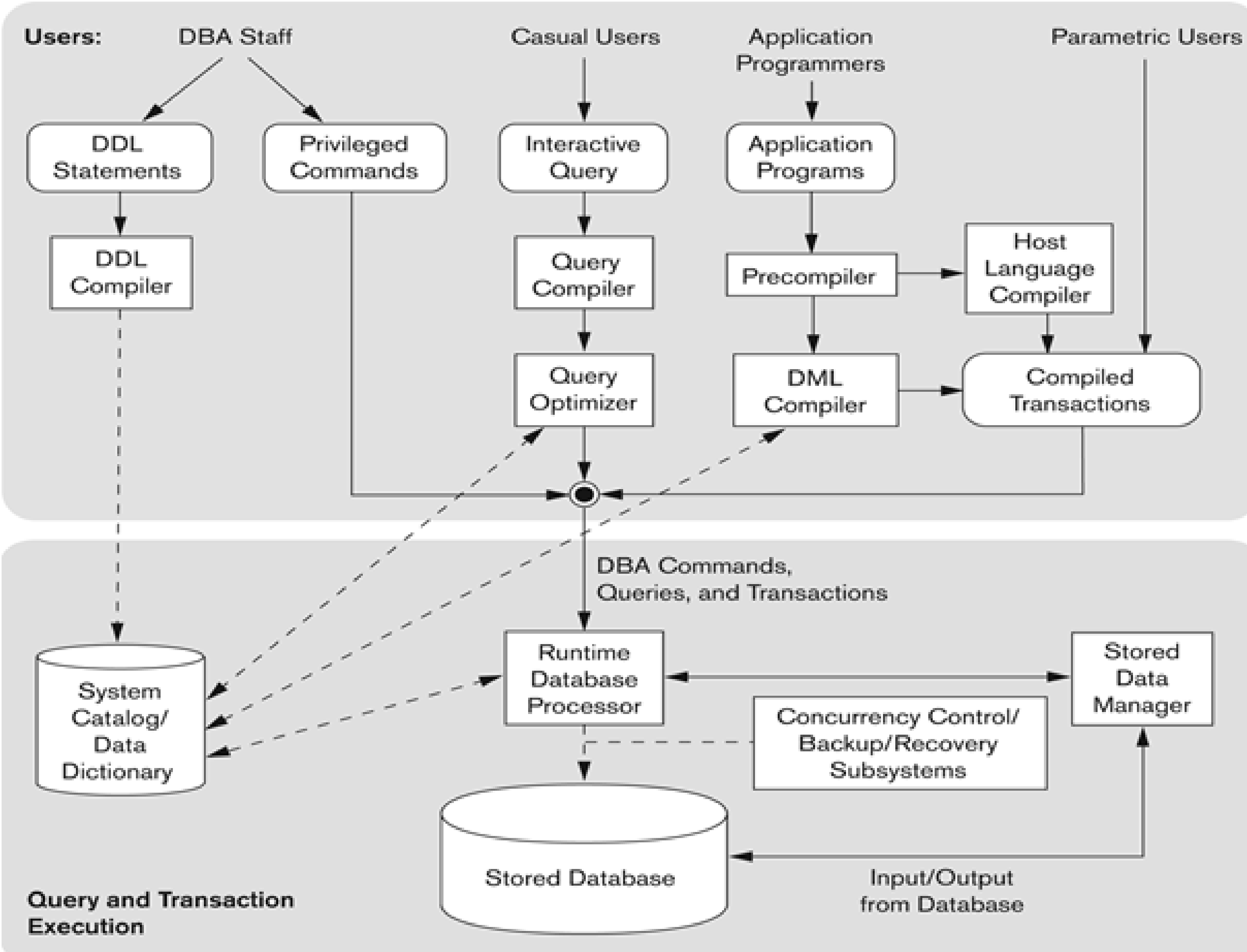
Database System Utilities

- 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.

Typical DBMS Component Modules



BREAK

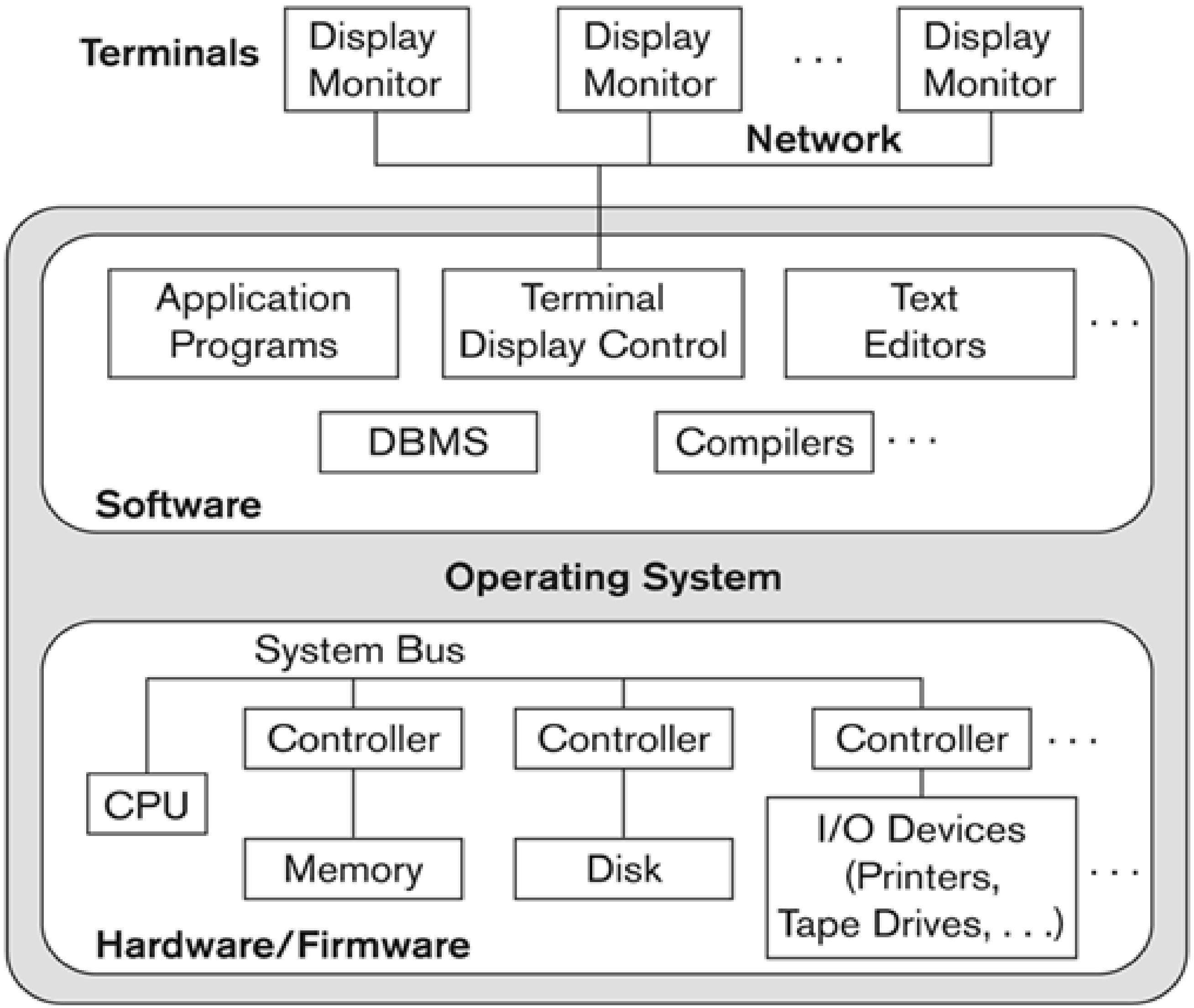
Outline

- Centralized and Client-Server Architectures
- Classification of DBMSs
- History of Data Models

Centralized and Client-Server DBMS Architectures

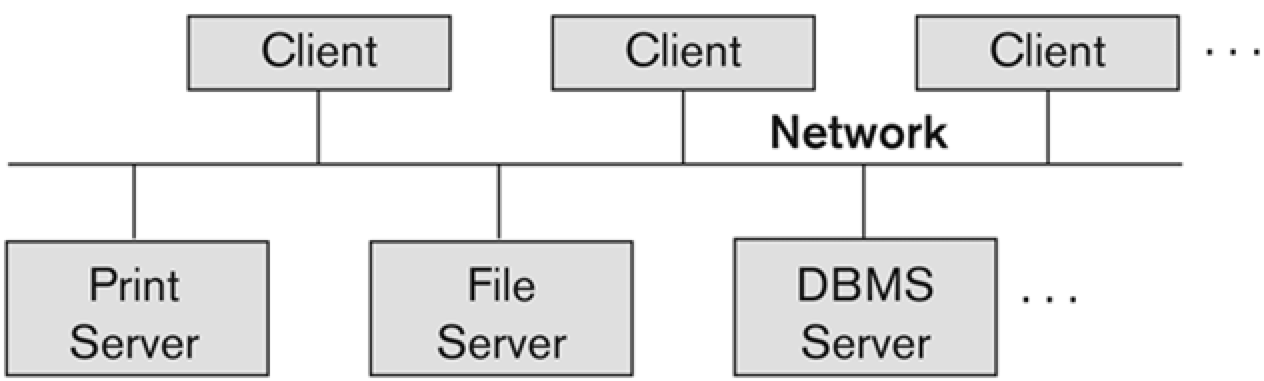
- Centralized DBMS:
  - Combines everything into single system including- DBMS software, hardware, application programs, and user interface processing software.
  - User can still connect through a remote terminal – however, all processing is done at centralized site.

A Physical Centralized Architecture



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

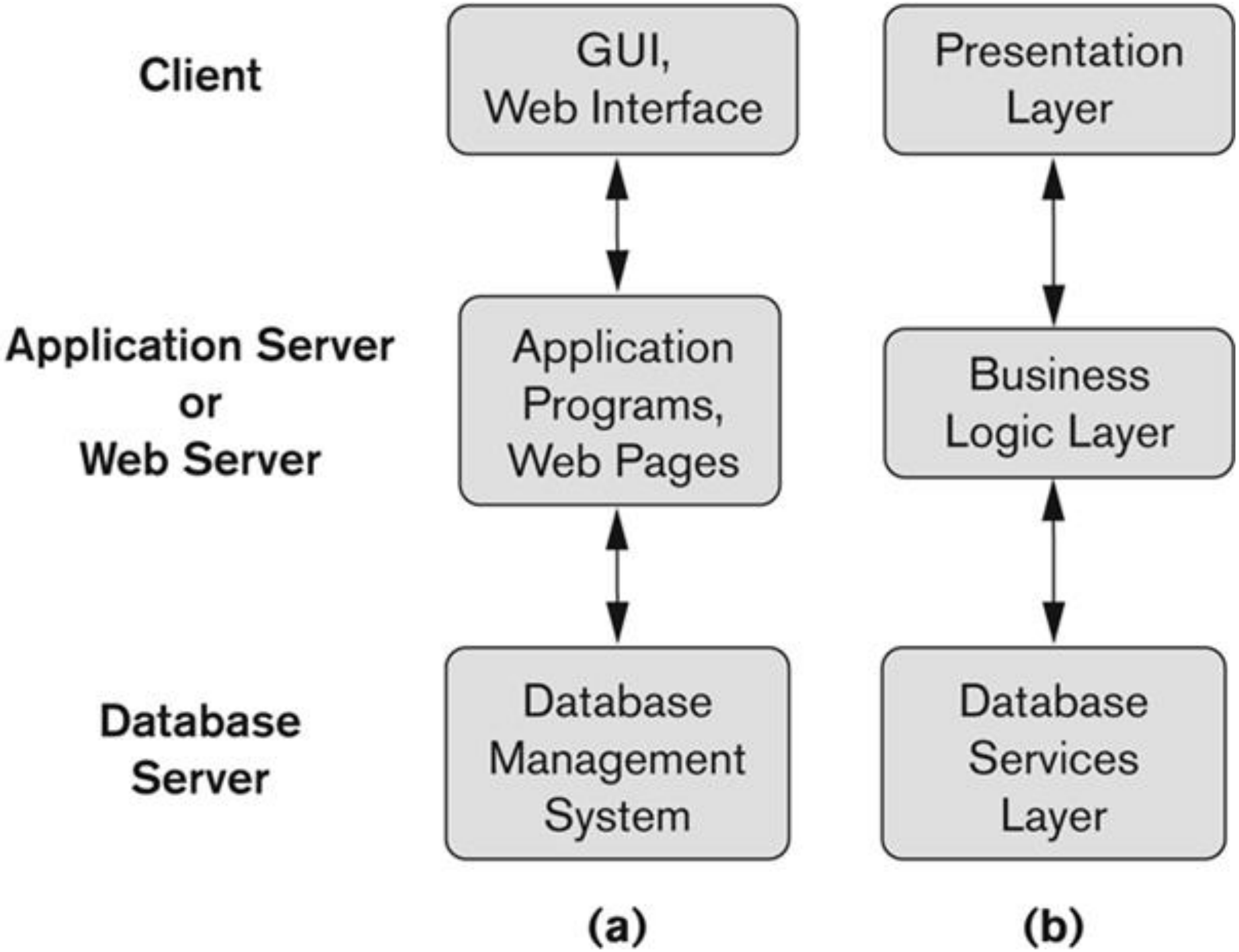
Two Tier Client-Server Architecture

- 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.
- Other variations of clients are possible: e.g., in some object DBMSs, more functionality is transferred to clients including data dictionary functions, optimization and recovery across multiple servers, etc.

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

Three-tier client-server architecture



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)

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.

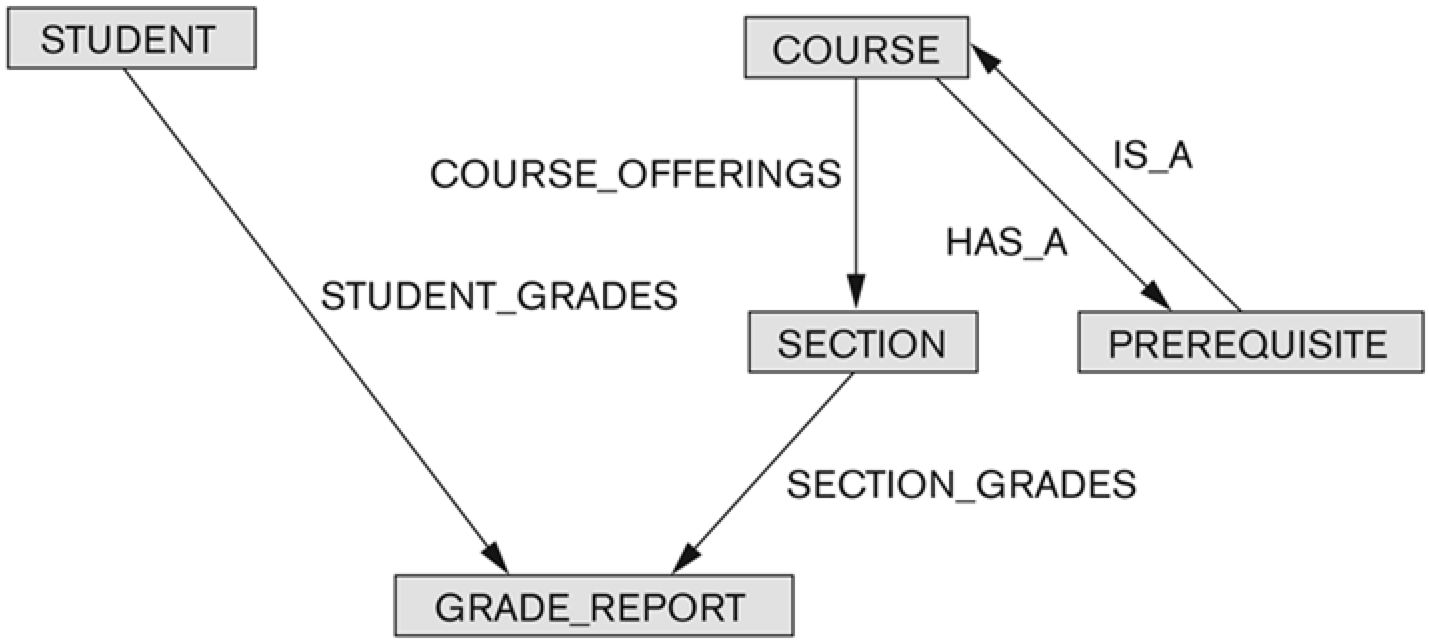
History of Data Models

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

History of Data 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.).

Example of Network Model Schema



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.

History of Data Models

- **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"

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 (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, etc.

History of Data Models

- **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.

History of Data Models

- **Object-Relational Models:**
  - Most Recent Trend. Started with Informix Universal Server.
  - Relational systems incorporate concepts from object databases leading to object-relational.
  - Exemplified in the latest versions of Oracle-10i, DB2, and SQL Server and other DBMSs.
  - Standards included in SQL-99 and expected to be enhanced in future SQL standards.

THANKS

References:
1. Silberschatz, H. Korth & S. Sudarshan, Database System Concepts, McGraw-Hill Education, 6th Edition, 2010.
2. R. Elmasri & S.B. Navathe, Fundamentals of Database Systems, Pearson Education, 6th edition, 2010.