Chapter 2

Database System Concepts and Architecture (from E&N and my editing)

- Data Models
  - Categories of Data Models
  - History of Data Models
- Schema
  - Three-Schema Architecture
- DBMS Component
- DBMS 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

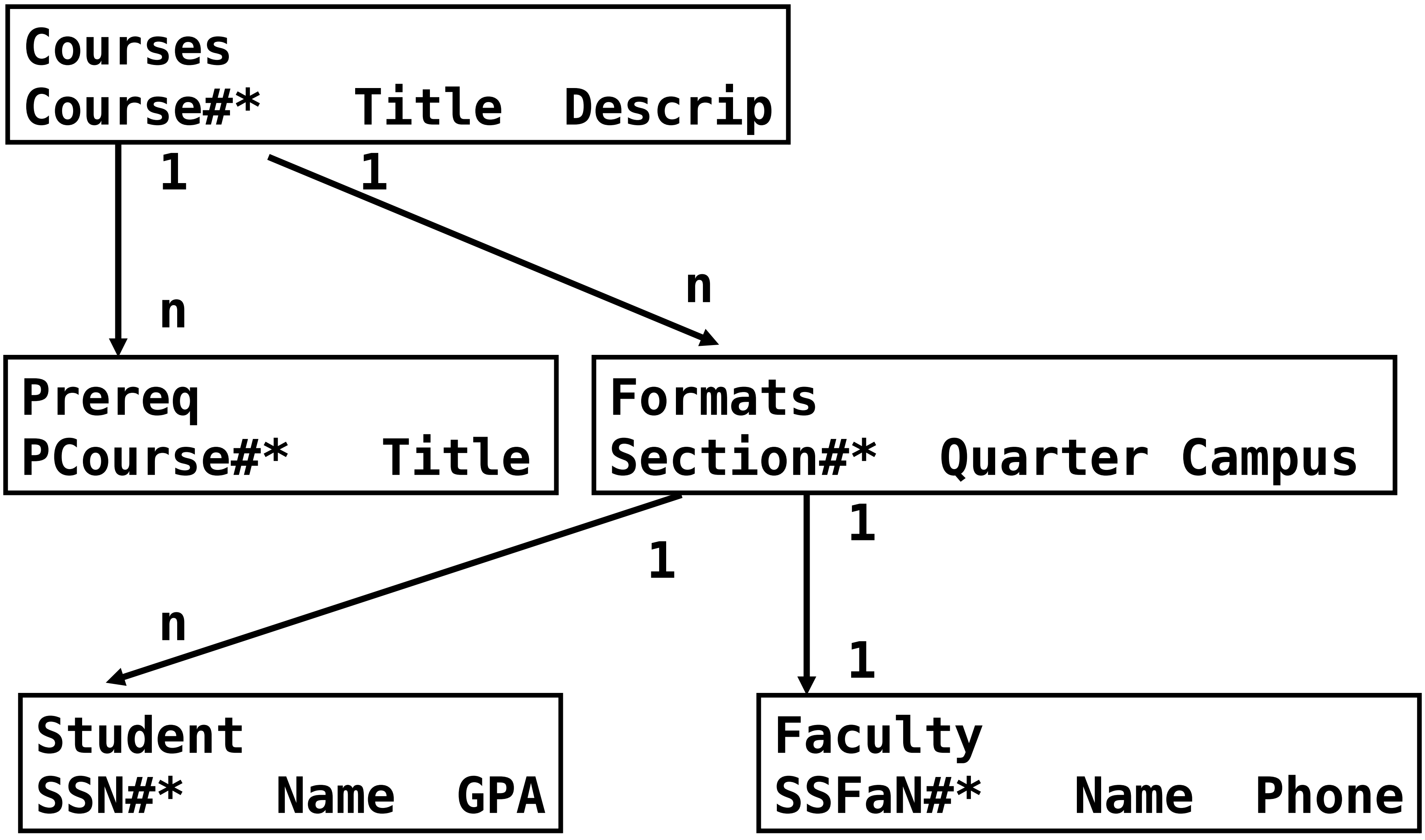
- Relational Model: proposed in 1970 by E.F. Codd (IBM), first commercial system in 1981-82. Now in several commercial products (DB2, ORACLE, SQL Server, SYBASE, INFORMIX).
- Network Model: the first one to be implemented by Honeywell in 1964-65 (IDS System). Adopted heavily due to the support by CODASYL (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 $O_{2,}$ 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.

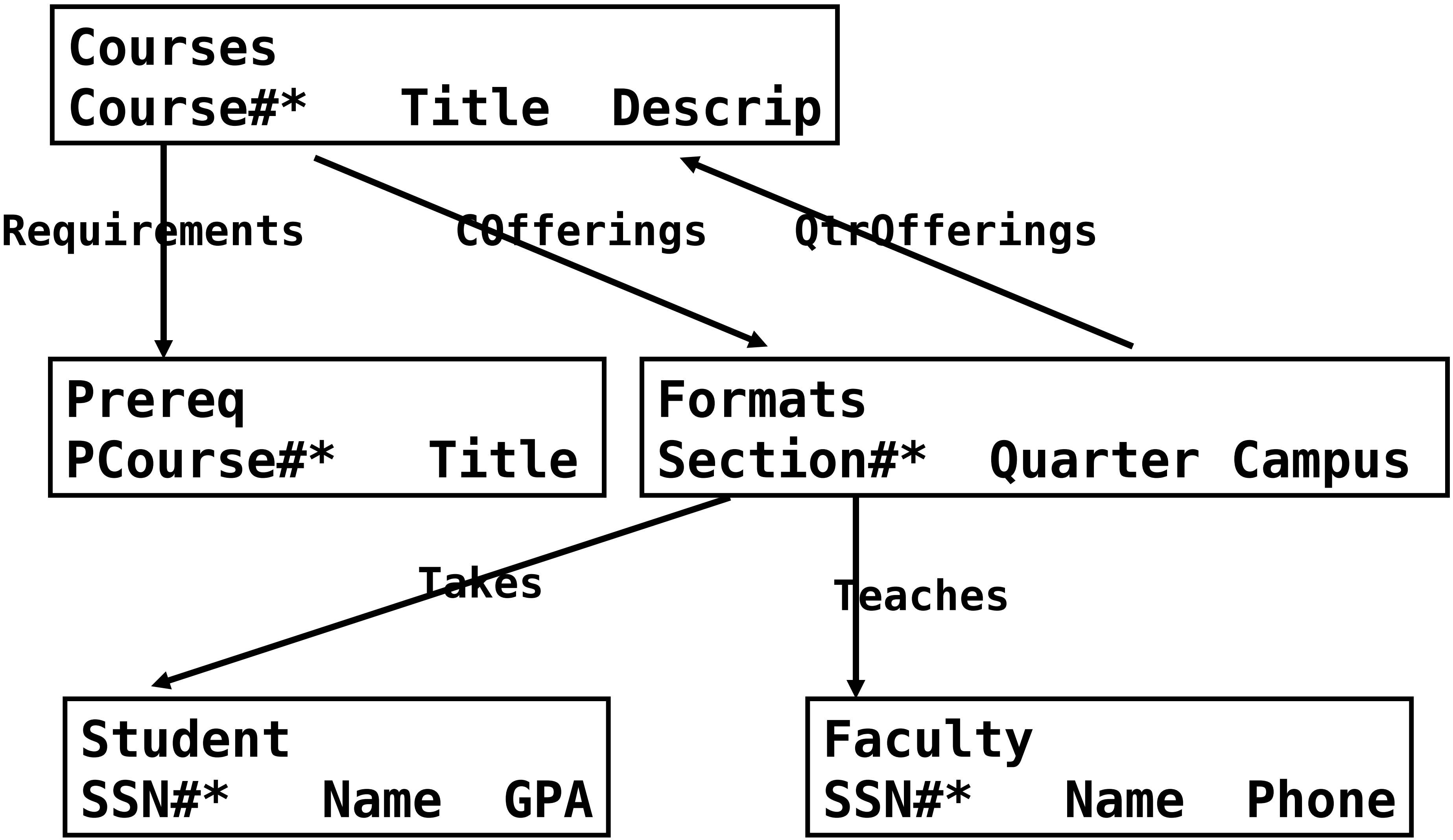
Hierarchical Model

- 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"


Courses
Course#* Title Descrip
1
1
n
n
Prereq
PCourse#* Title
Formats
Section#* Quarter Campus
1
1
n
1
Student
SSN#* Name GPA
Faculty
SSFaN#* Name Phone

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”


Courses
Course#* Title Descrip
Requirements
COfferings
QtrOfferings
Prereq
PCourse#* Title
Formats
Section#* Quarter Campus
Takes
Teaches
Student
SSN#* Name GPA
Faculty
SSN#* Name Phone

Relational Model

- Relational Model of Data Based on the Concept of a Relation
- Relation- a Mathematical Concept Based on Sets
- Strength of the Relational Approach to Data Management Comes From the Formal Foundation Provided by the Theory of Relations
- RELATION: A Table of Values
  - A **Relation** May Be Thought of as a **Set** of **Rows**
  - A Relation May Alternately be Though of as a **Set** of **Columns**
  - Each **Row** of the Relation May Be Given an **Identifier**
  - Each **Column** Typically is Called by its Column Name or Column Header or Attribute Name

Relational Tables - Rows/Columns/Tuples

STUDENT	Name	StudentNumber	Class	Major
	Smith	17	1	CS
	Brown	8	2	CS

COURSE	CourseName	CourseNumber	CreditHours	Department
	Intro to Computer Science	CS1310	4	CS
	Data Structures	CS3320	4	CS
	Discrete Mathematics	MATH2410	3	MATH
	Database	CS3380	[illegible]	[illegible]

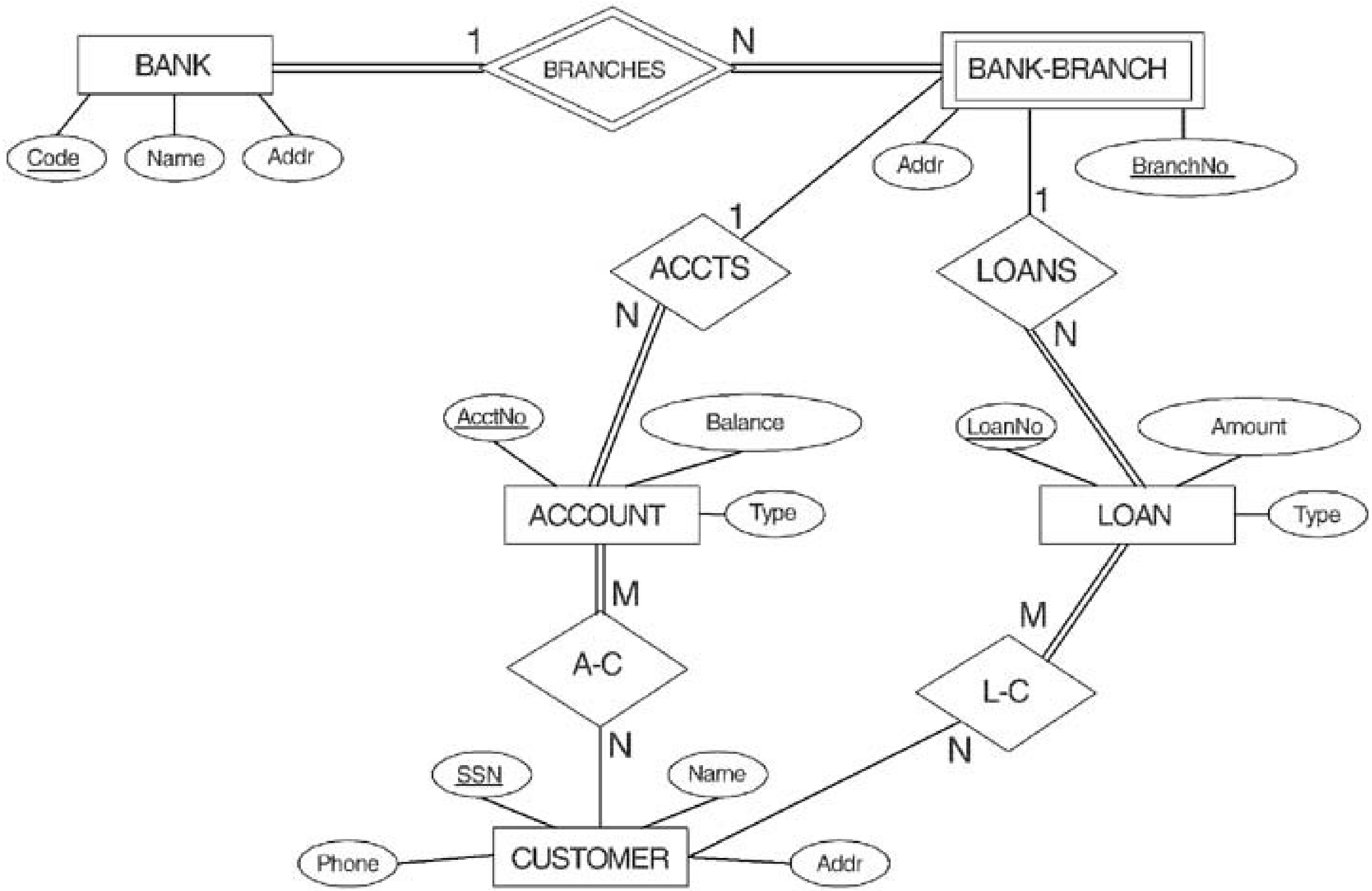
SECTION	SectionIdentifier	CourseNumber	S
	85	MATH2410	
	92	CS1310	
	102	CS3320	
	112	MATH2410	
	119	CS1310	
	135	CS3380	

GRADE_REPORT	StudentNumber	SectionIdentifier	Grade
	17	112	B
	17	119	C
	8	85	A
	8	92	A
	8	102	B
	8	135	A

PREREQUISITE	CourseNumber	PrerequisiteNumber
	CS3380	CS3320
	CS3380	MATH2410
	CS3320	CS1310

Entity Relationship (ER) Data Model

- Originally Proposed by P. Chen, ACM TODS, Vol. 1, No. 1, March1976
- Conceptual Modeling of Database Requirements
- Allows an Application's Information to be Characterized
- Basic Building Blocks are Entities and Relationships
- Well-Understood and Studied Technique
- Well-Suited for Relational Database Development
- Did Not Originally Include Inheritance!!


BANK
1
BRANCHES
N
BANK-BRANCH
Code
Name
Addr
Addr
BranchNo
1
1
ACCTS
LOANS
N
N
AcctNo
Balance
LoanNo
Amount
ACCOUNT
Type
LOAN
Type
M
M
A-C
L-C
N
N
SSN
Name
Phone
CUSTOMER
Addr

Object-Oriented Database Models/Systems

- Reasons for Creation of Object Oriented Databases
  - Need for More Complex Applications
  - Need for Additional Data Modeling Features
  - Increased Use of Object-oriented Programming Languages
- Experimental Systems: Orion at MCC, IRIS at H-P Labs, Open-oodb at T.I., ODE at ATT Bell Labs, Postgres - Montage - Illustra at UC/B, Encore/observer at Brown
- Commercial OO Database Products: Ontos, Gemstone (-> Ardent), Objectivity, Objectstore (-> Excelon), Versant, Poet, Jasmine (Fujitsu – GM)
- Also - Relational Products with Object Capabilities

Object-Oriented Database Models/Systems

- OO Databases Try to Maintain a Direct Correspondence Between Real-world and DB Objects
- Object have State (Value) and Behavior (Operations)
  - In OO Databases
    - Objects May Have an Object Structure of Arbitrary Complexity in Order to Contain All of the Necessary Information That Describes the Object
  - In Traditional Database Systems
    - Information About a Complex Object is Often Scattered Over Many Relations or Records
    - Leads to Loss of Direct Correspondence Between a Real-world Object and Its Database Representation
- Supports OO Programming Concepts: Inheritance, Polymorphism, etc.

Object-Oriented Database Declarations

- Specifying the Object Types Employee, Date, and Department Using Type Constructors

```
define type Employee:
    tuple  (        fname:          string;
                    minit:          char;
                    lname:          string;
                    ssn:            string;
                    birthdate:      Date;
                    address:        string;
                    sex:            char;
                    salary:         float;
                    supervisor:     Employee;
                    dept:           Department;        );
define type Date
    tuple  (        year:           integer;
                    month:          integer;
                    day:            integer;    );
define type Department
    tuple  (        dname:          string;
                    dnumber:        integer;
                    mgr:            tuple (   manager:    Employee;
                                              startdate:  Date;        );
                    locations:      set(string);
                    employees:      set(Employee);
                    projects        set(Project);    );
```

- Adding Operations to Definitions of Employee and Department:

```
define class DepartmentSet:
    type          set(Department);
    operations  add_dept(d: Department): boolean;
            (* adds a department to the DepartmentSet object *)
                  remove_dept(d: Department): boolean;
            (* removes a department from the DepartmentSet object *)
                  create_dept_set:          DepartmentSet;
                  destroy_dept_set:         boolean;
end DepartmentSet;

    ...

persistent name AllDepartments: DepartmentSet;
(* AllDepartments is a persistent named object of type DepartmentSet *)

    ...

d:= create_dept;
(* create a new Department object in the variable d *)

    ...

b:= AllDepartments.add_dept(d);
(* make d persistent by adding it to the persistent set AllDepartments *)

    ...
```

Schemas

- **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 State/Snapshot**: The actual data stored in a database at a *particular moment in time*. Also called the **current set of occurrences/instances**).

STUDENT

Name	StudentNumber	Class	Major

COURSE

CourseName	CourseNumber	CreditHours	Department

PREREQUISITE

CourseNumber	PrerequisiteNumber

SECTION

SectionIdentifier	CourseNumber	Semester	Year	Instructor

GRADE_REPORT

StudentNumber	SectionIdentifier	Grade

Schemas versus Instances

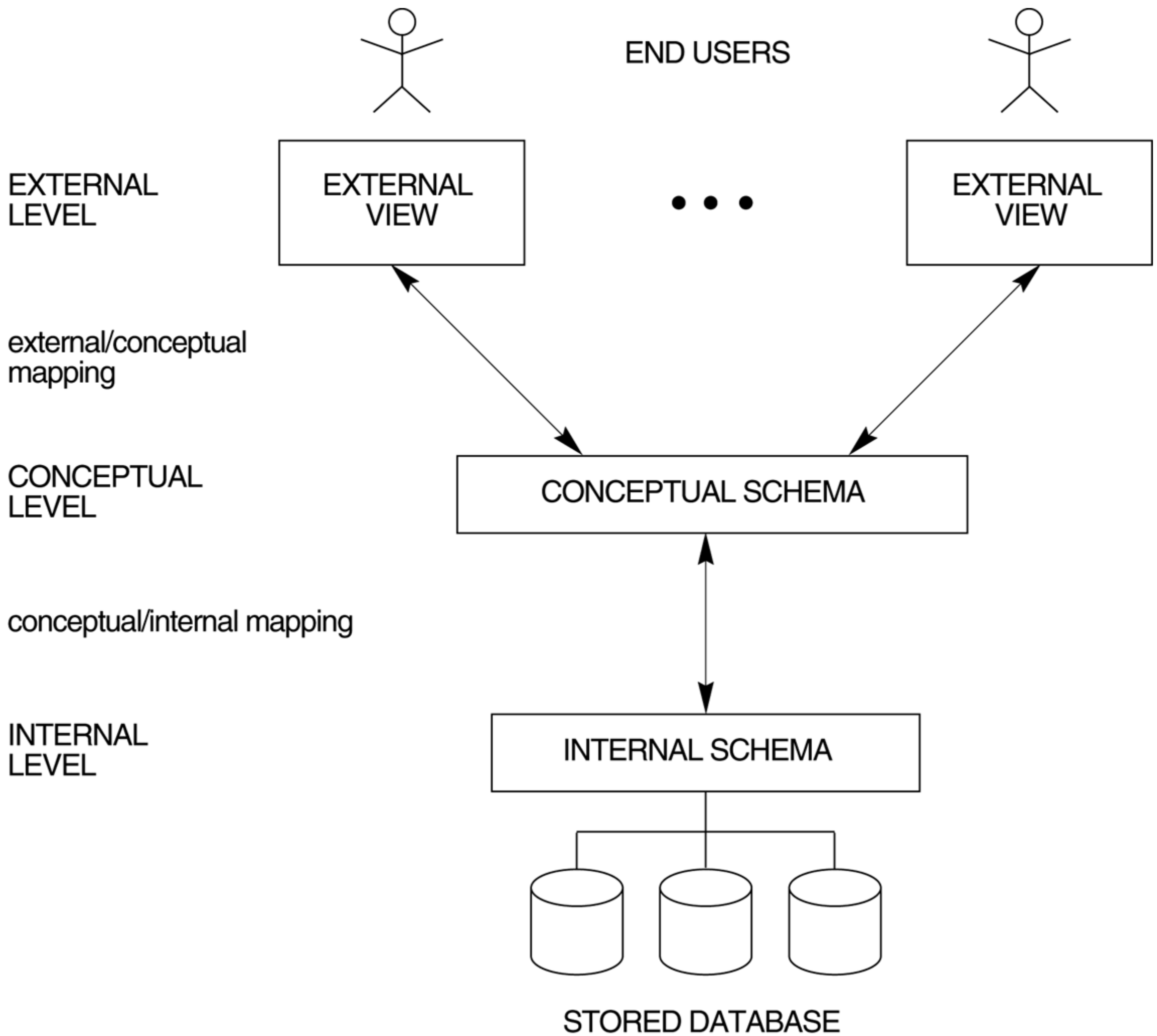
- **Database Schema**: The ***description*** of a database. Includes descriptions of the database structure and the constraints that should hold on the database.
- **Database Instance**: The **actual data stored** in a database at a *particular moment in time*. Also called **database state** (or **occurrence**).

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.

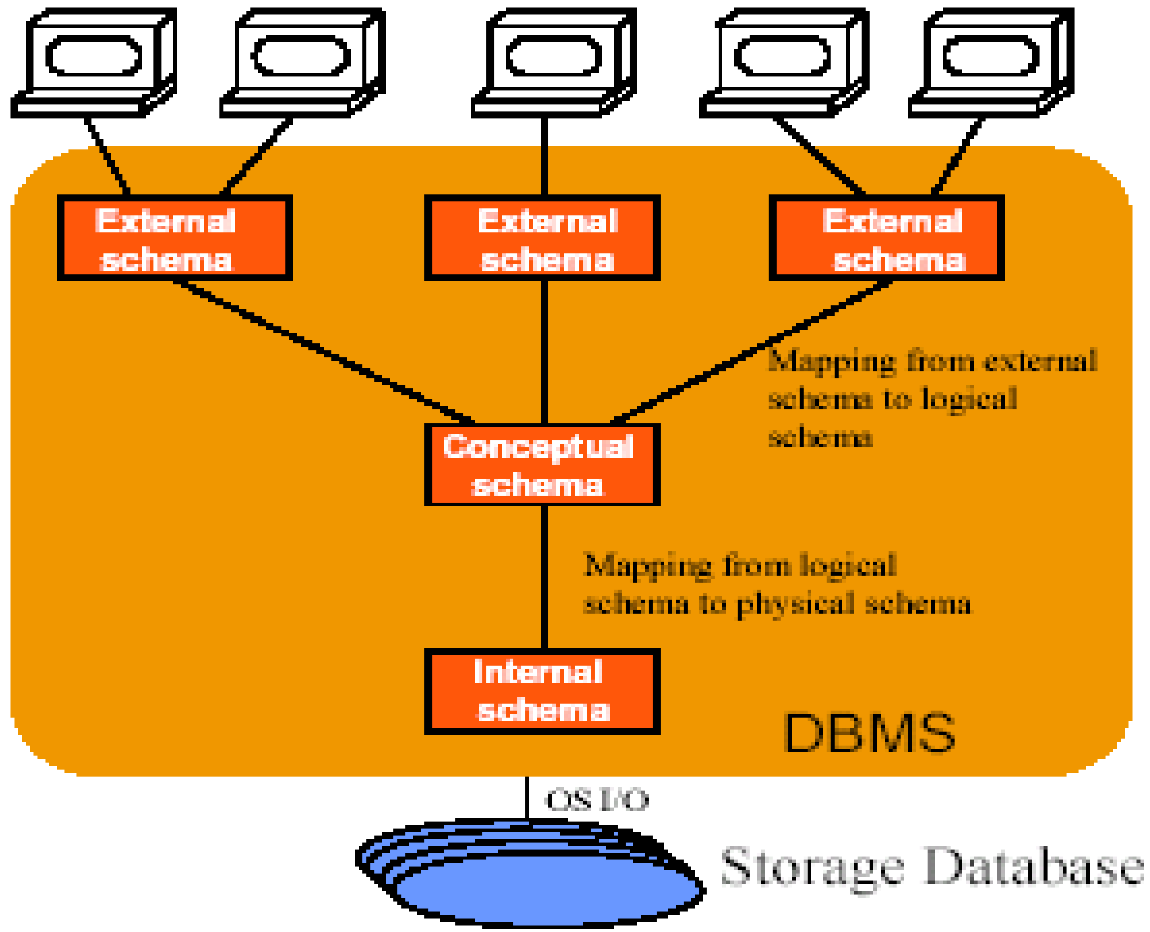

END USERS
EXTERNAL LEVEL
EXTERNAL VIEW
EXTERNAL VIEW
external/conceptual mapping
CONCEPTUAL LEVEL
CONCEPTUAL SCHEMA
conceptual/internal mapping
INTERNAL LEVEL
INTERNAL SCHEMA
STORED DATABASE

Users

User's view on the **use of data**

Logical schema: the **meaning of data**

Physical schema: the **storage of 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.

Conceptual Schema

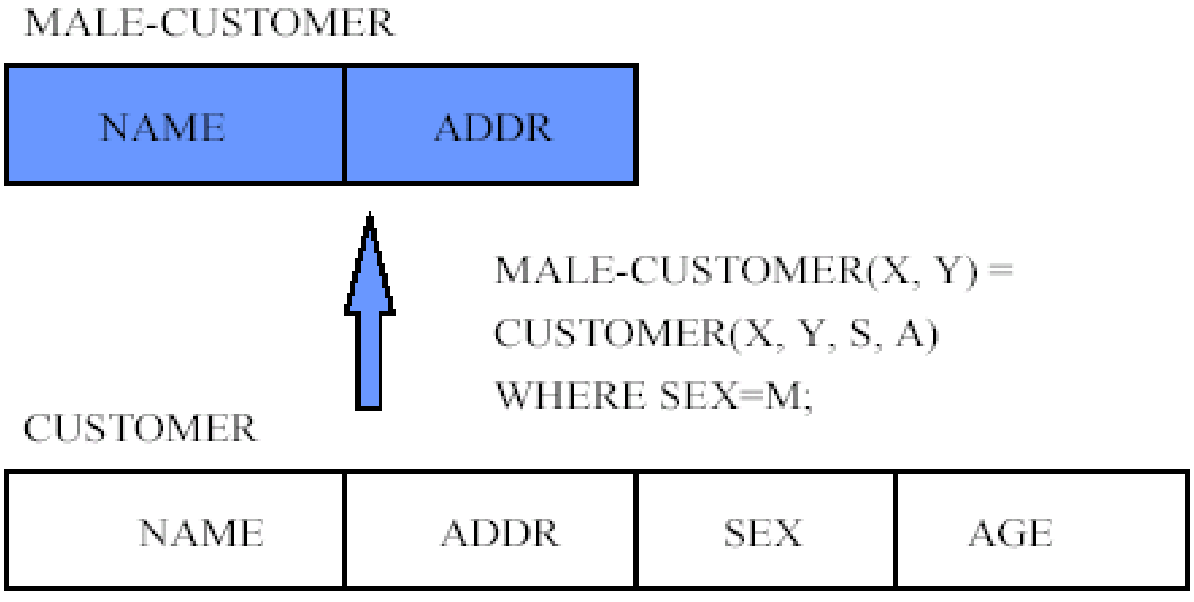
- Describes the Meaning of Data in the Universe of Discourse
  - Emphasizes on General, Conceptually Relevant, and Often Time Invariant Structural Aspects of the Universe of Discourse
- Excludes the Physical Organization and Access Aspects of the Data

CUSTOMER

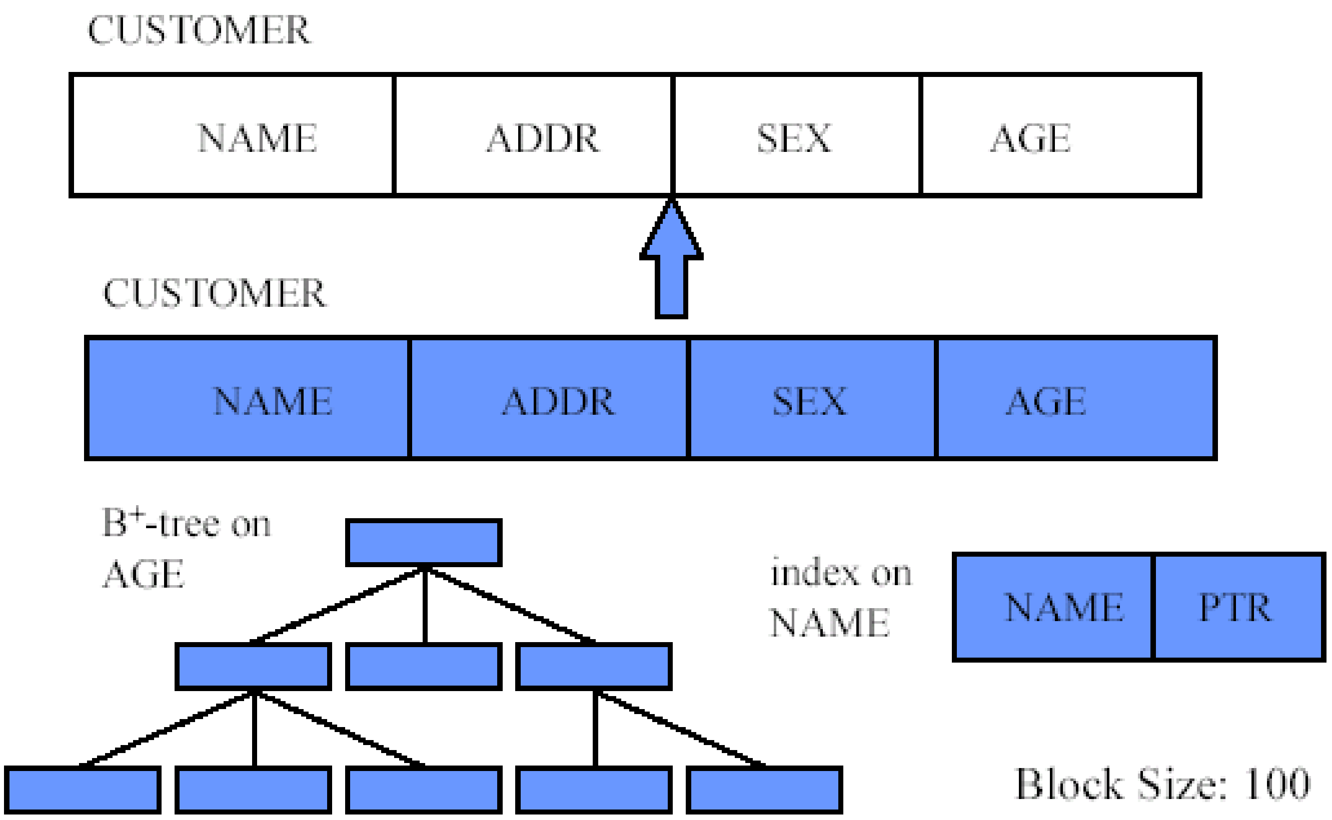
NAME	ADDR	SEX	AGE

Ext Schema

- Describes Parts of the Information in the Conceptual Schema in a form Convenient to a Particular User Group's View
- Derived from the Conceptual Schema



Internal Schema



Unified Example of Three Schemas

An Example Query:

"List all employees whose has more than 5 years working experience?"

SELECT e.ENAME, e.DEPT, e.EXP
FROM EMP e
WHERE e.EXP > 5 year.

External Schema:

CREATE EMP(ENAME, DEPT, EXP)
AS VIEW OF EMPLOYEE(EN, DNO, EXP_YEAR)

CREATE PAYROLL(EN, SAL, SSN, BirthDate)
AS VIEW OF EMPLOYEE(SSN,EN,SALARY,BDATE)

Conceptual Schema:

EMPLOYEE(SSN, EN, DNO, SALARY, EXP_YEAR, BDATE, STARTDATE)

Internal Schema:

Cluster Index on SNN;

No-cluster B-tree Indexes on DNO, EXP_YEAR, STARTDATE.

Data Independence

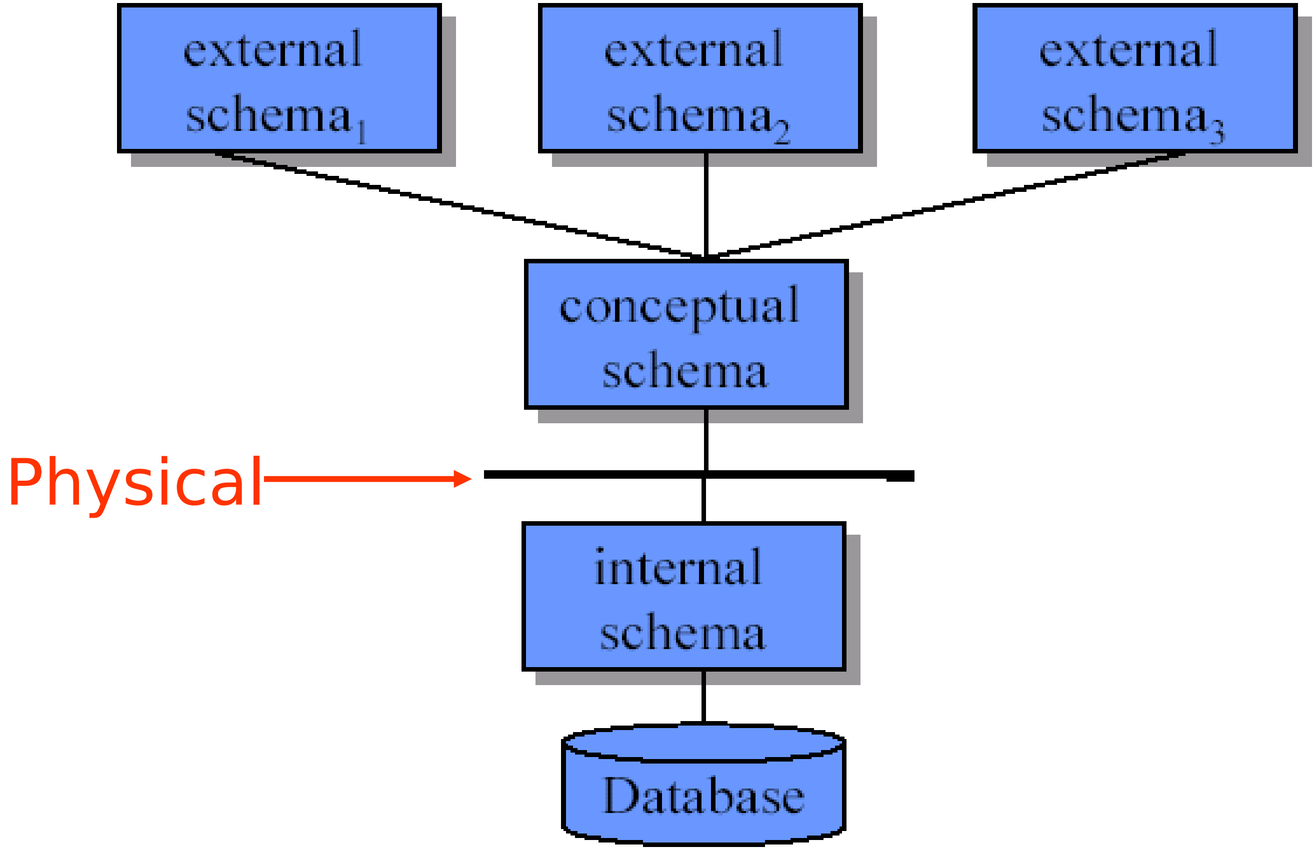
- **Logical Data Independence**: The capacity to change the conceptual schema without having to change the external schemas and their application programs.
- **Physical Data Independence**: The capacity to change the internal schema without having to change the conceptual schema.

- Ability that Allows Application Programs Not Being Affected by Changes in Irrelevant Parts of the Conceptual Data Representation, Data Storage Structure and Data Access Methods
- Invisibility (Transparency) of the Details of Entire Database Organization, Storage Structure and Access Strategy to the Users
  - Both Logical and Physical
- Recall Software Engineering Concepts:
  - ***Abstraction*** the Details of an Application's Components Can Be Hidden, Providing a Broad Perspective on the Design
  - ***Representation Independence***: Changes Can Be Made to the Implementation that have No Impact on the Interface and Its Users

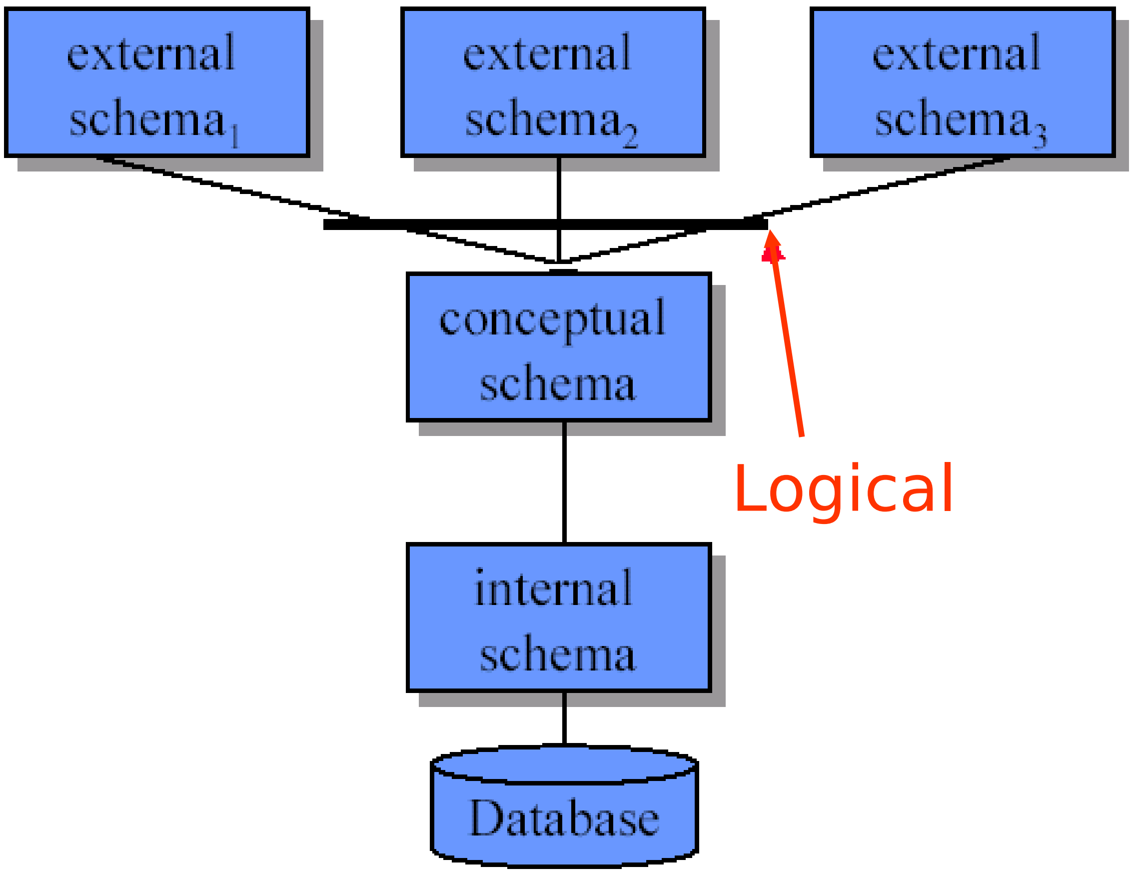
Data Independence

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.

Physical Data Independence



Logical Data Independence



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

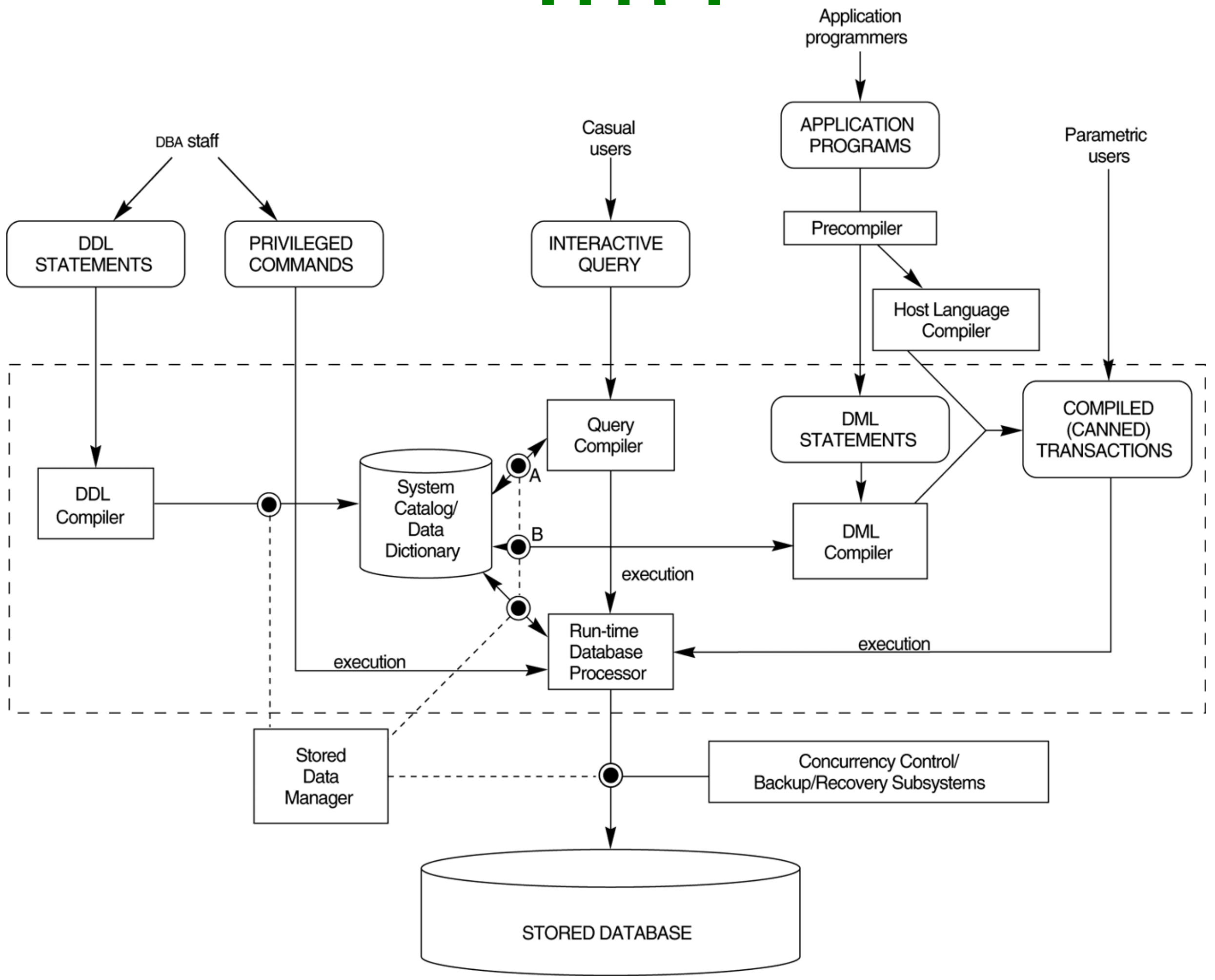
Database System Env

- Main DBMS Modules
  - DDL Compiler
  - DML Compiler
  - Ad-hoc (Interactive) Query Compiler
  - Run-time Database Processor
  - Stored Data Manager
  - Concurrency/Back-Up/Recovery Subsystem
- DBMS Utility Modules
  - Loading Routines
  - Backup Utility
  - …

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

Component Moduls and Intr



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), VB, Java, C, C++, Ms. Visio, ER-Win, DBDesigner

Centralized and Client-Server Architectures

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

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

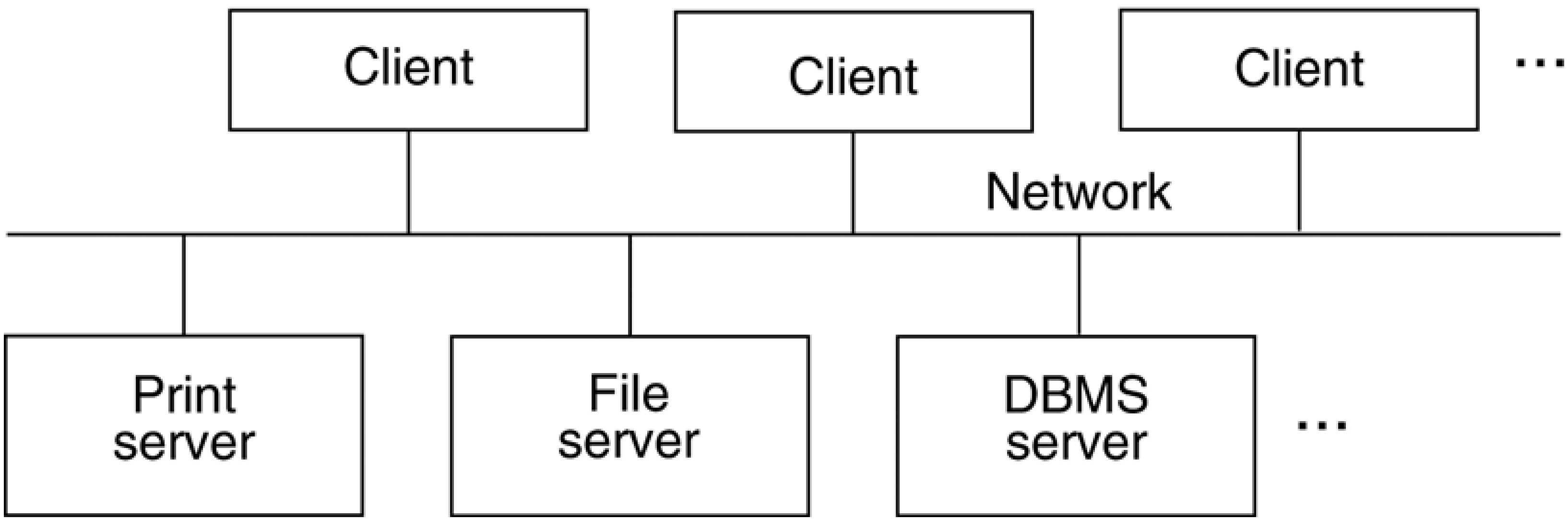
Two Tier Client-Server Architecture

- **User Interface Programs and Application Programs** run on the client side
- Interface called **ODBC (Open Database Connectivity** – see 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**.

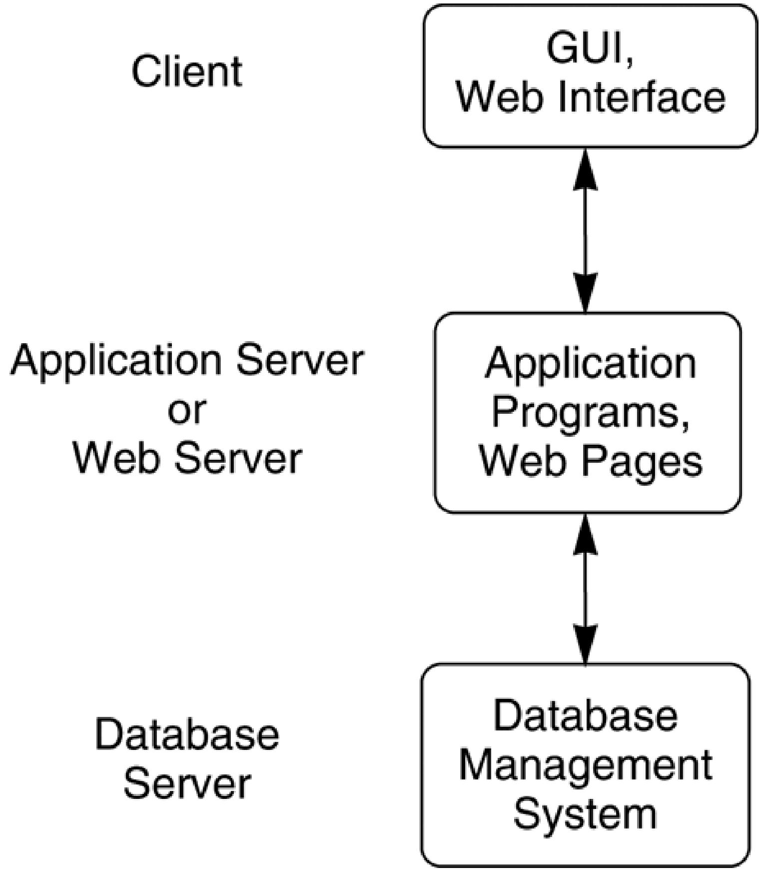
Logical Two Tier/Client Server Architecture



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

Logical Three Tier



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.*

Variations of Distributed Environments:

- Homogeneous DDBMS
- Heterogeneous DDBMS
- Federated or Multidatabase Systems

According to the data models	• object-oriented DBMS (ObjectStore, Ontos, etc.) • **relational DBMS** (Oracle, Sybase, Informix, DB2, Microsoft SQL server etc.) • network DBMS (DBTG ...) • hierarchical DBMS (IMS ...)
According to the number of users	• single-user DBMS (mainly for PCs) • multi-user DBMS
According to the number of sites	• centralized DBMS (Oracle, Sybase, etc.) • distributed DBMS (R*, ...) • federated DBMS homogeneous DBS heterogeneous DBS
According to the types of access methods	• general purpose DBMS • special purpose DBMS