Chapter 2

Database Systems Concepts and Architecture

Data Abstraction

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

- By data abstraction we mean that the system hides certain details of how the data are physically stored and maintained

- Data abstraction enables different users to perceive data at their preferred level of detail

- Data model provides means to achieve data abstraction

Data Model

- Data model is a collection of concepts that 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 updates on the database

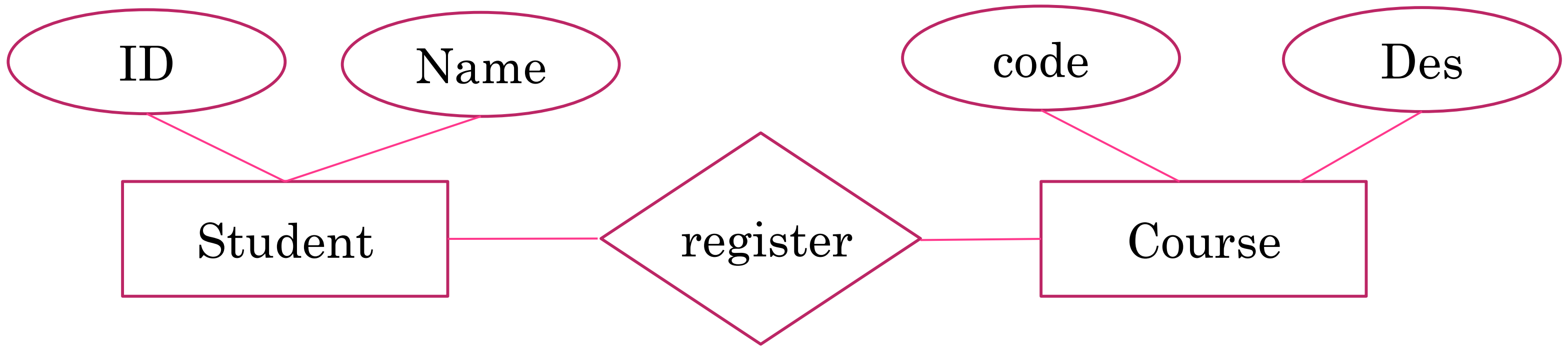
- There are different categories of data models

Data Model Categories

- Many data models have been proposed, which we can categorize according to the types of concepts they use to describe the database structure.
  - Conceptual data model: describes the database structure in a way that most users can understand
  - Representational data model: describes the database structure in a way that will be implemented in DBMS
  - Physical data model: describes how the database will be physically stored on the computer storage media typically magnetic disks

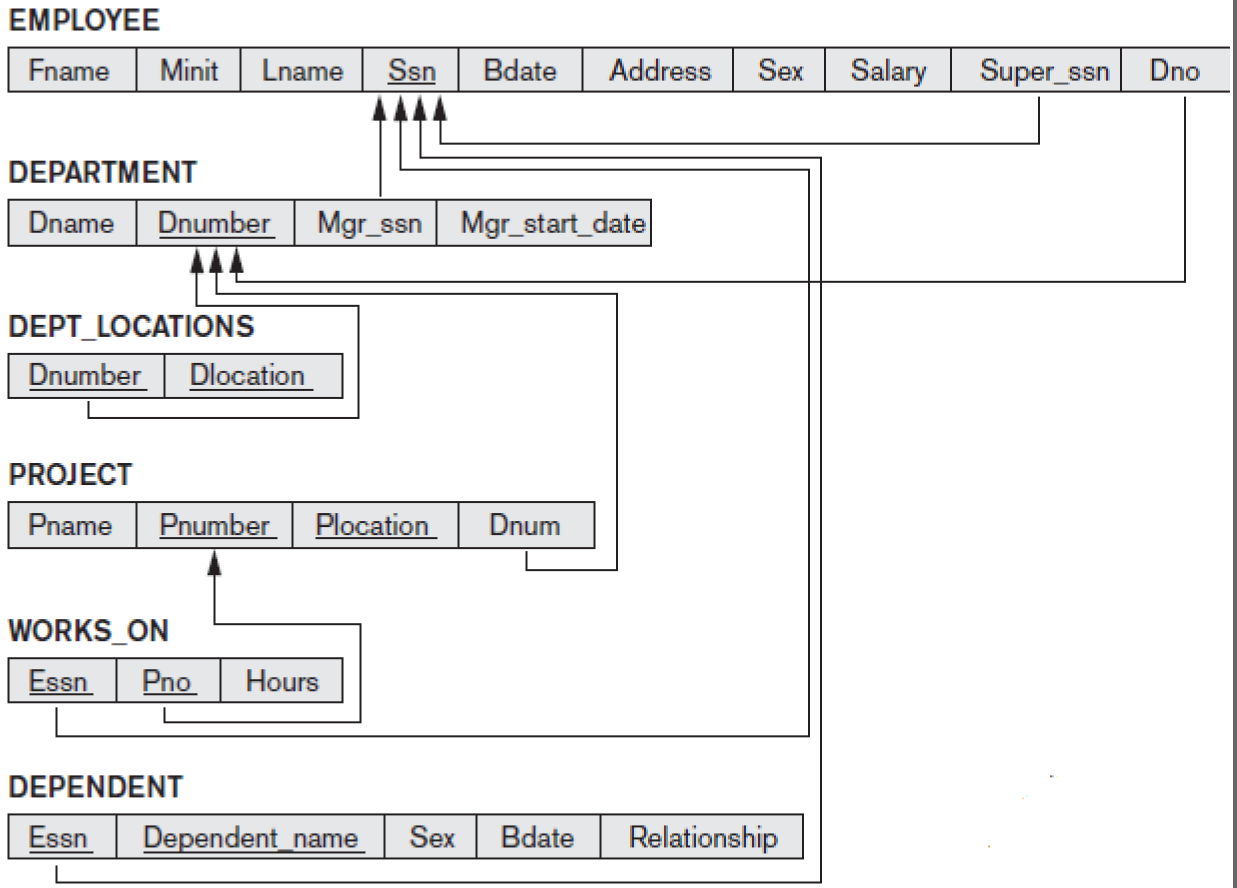
Conceptual Data Model

- It uses concepts such as entities, attributes and relationships
- An ***entity*** represents a real-world object or concept, such as an course or a student from the miniworld that is described in the database.
- An ***attribute*** represents data of interest that further describes an entity, such as the student's name or address.
- A ***relationship*** among two or more entities represents an association among the entities
- Entity relationship model (ER) is a popular example of this model



Representational Data Model

- It is the type of models used most frequently in traditional commercial DBMSs.
- These include the widely used **relational data model**, as well the **network** and **hierarchical models**
- It represents data by using tables and record structures and hence are sometimes called **record-based data models**



EMPLOYEE
Fname | Minit | Lname | Ssn | Bdate | Address | Sex | Salary | Super_ssn | Dno
DEPARTMENT
Dname | Dnumber | Mgr_ssn | Mgr_start_date
DEPT_LOCATIONS
Dnumber | Dlocation
PROJECT
Pname | Pnumber | Plocation | Dnum
WORKS_ON
Essn | Pno | Hours
DEPENDENT
Essn | Dependent_name | Sex | Bdate | Relationship

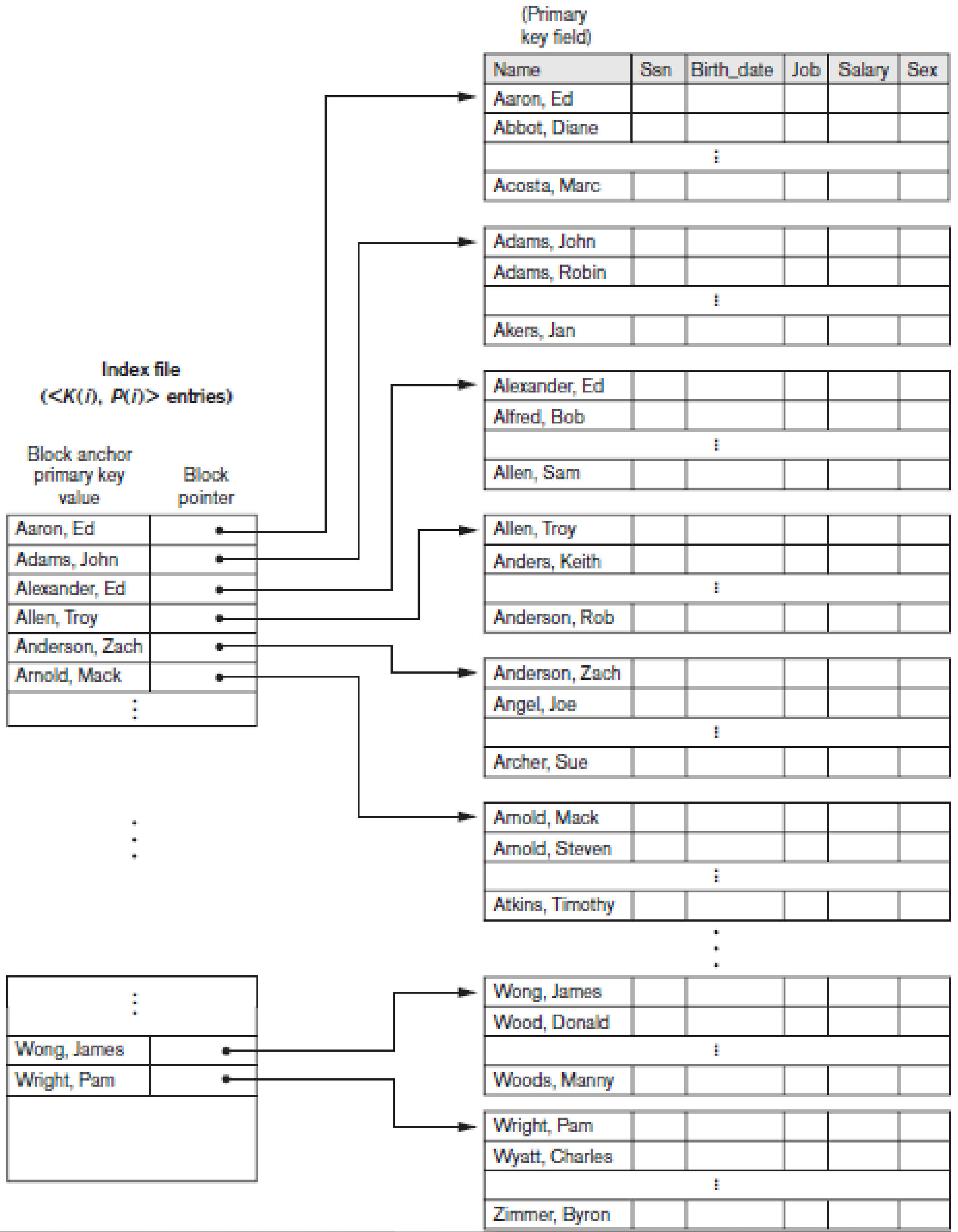
Physical Data Model

- Physical data models describe how data is stored as files in the computer by representing information such as where data is stored record formats, record orderings, and access paths.

- An **access path** is a structure that makes the search for particular database records efficient.

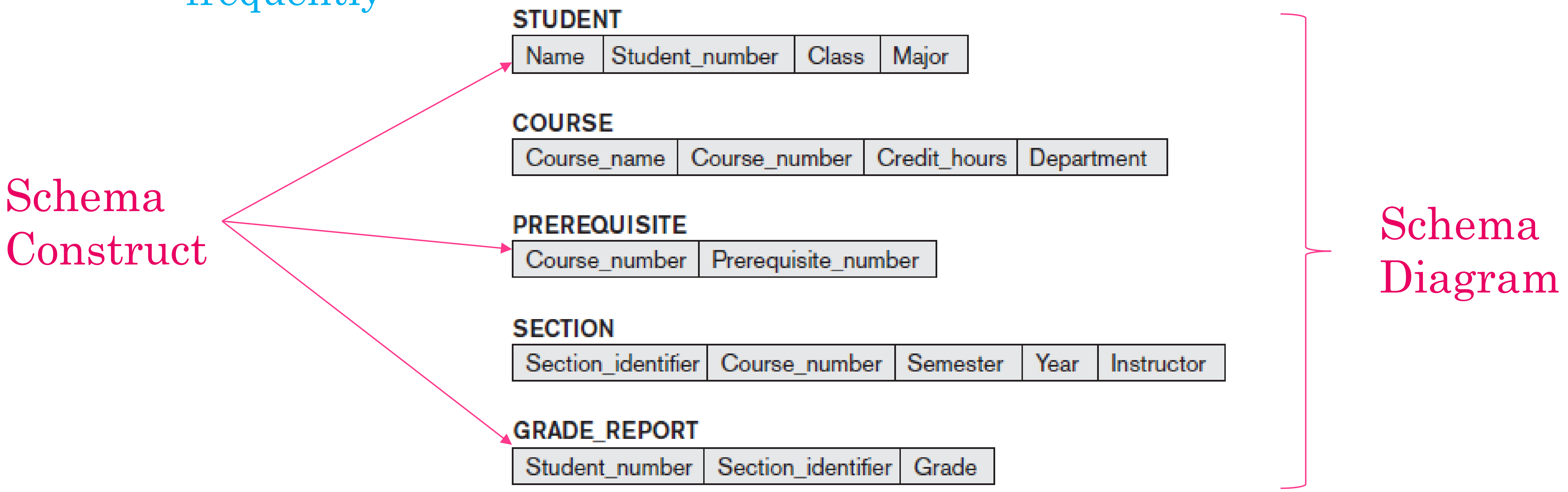
- An **index** is an example of an access path that allows direct access to data using an index term or a keyword. It is similar to the index of book

Example Of Access Path: Index



Schemas, Instances, and Database State

- In any 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



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, Instances, and Database State

- 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
- The actual data in a database may change quite frequently
- The database changes every time we add a new student or enter a new grade.
- The data in the database at a particular moment in time is called a **database state** or **snapshot**. It is also called the *current* set of **occurrences** or **instances**
- 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.

Schemas, Instances, and Database State

- The distinction between database schema and database state is very important.
- When we **define** a new database, we specify its database schema.
- At this point, the corresponding database state is the *empty state* with no data. 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 an update operation is applied to the database, we get another database state. At any point in time, the database has a *current 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

Three Schema Architecture and Data Independence

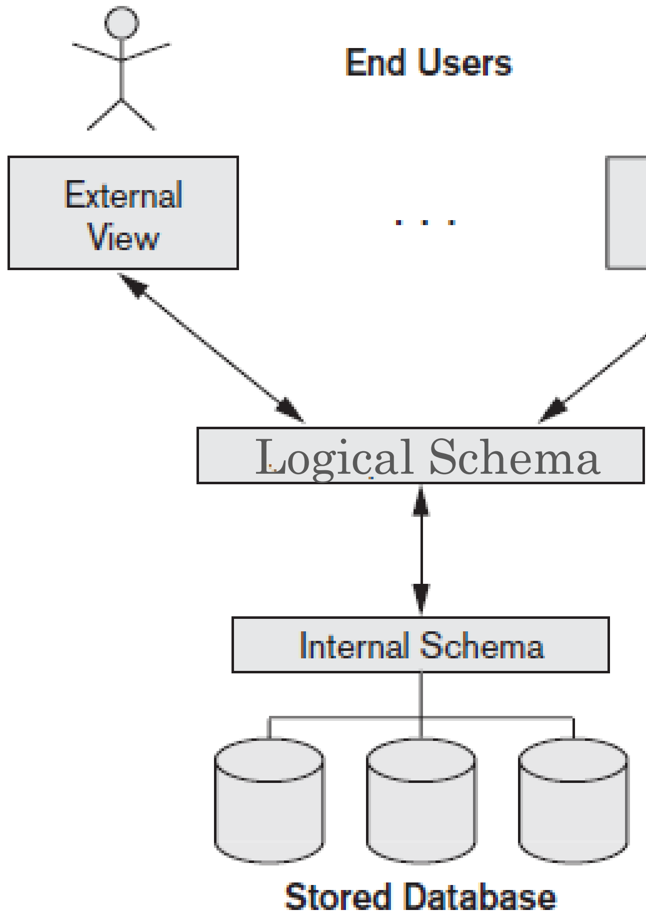
- Three important characteristics of the database approach, are
- (1) use of a catalog to store the database description (schema) so as to make it self-describing
- (2) insulation of programs and data (data independence)
- (3) support of multiple user views.
- The **three-schema architecture** is proposed to help achieve and visualize these characteristics

Three Schema Architecture and Data Independence

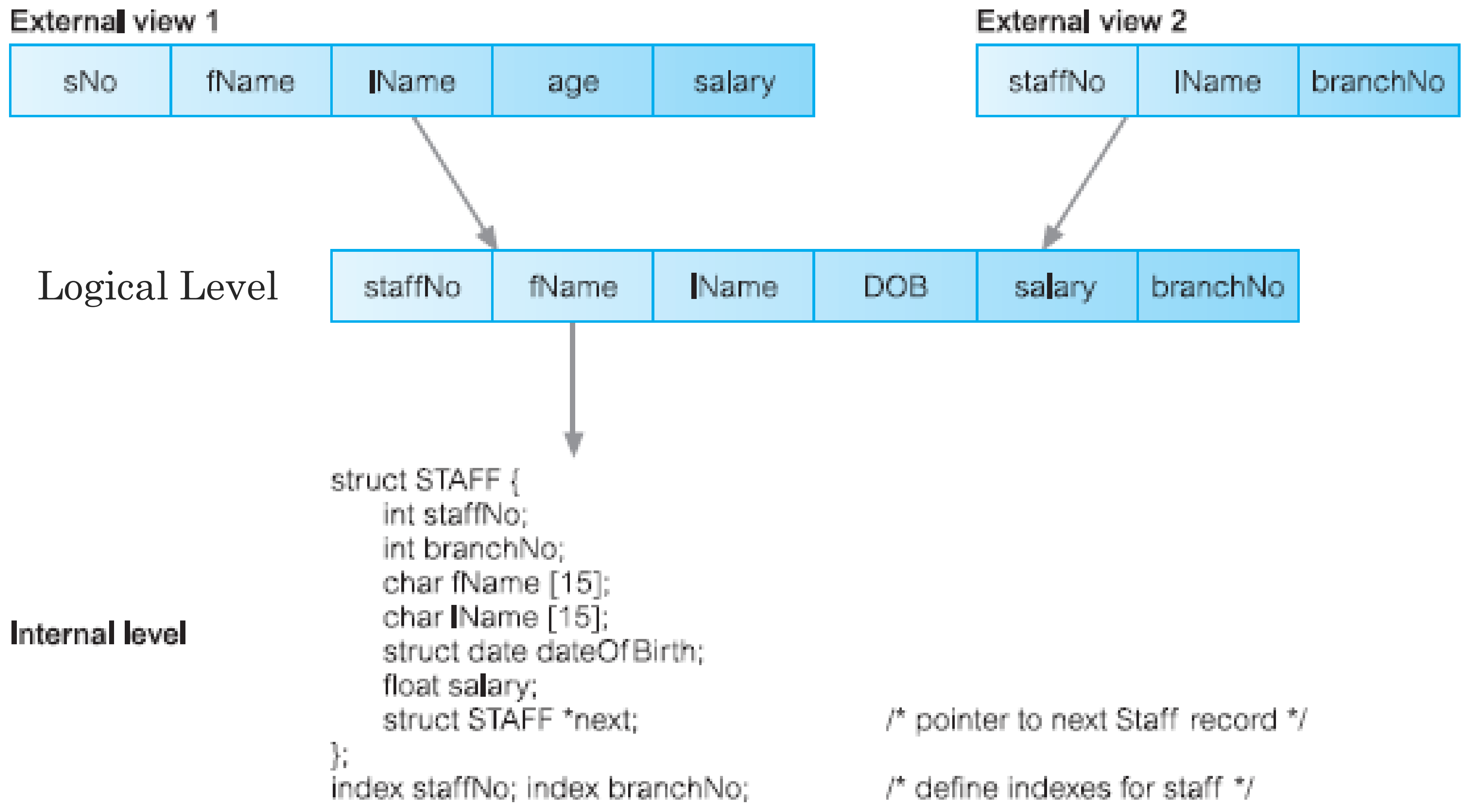
External Level

Logical Level

Internal Level



Three Schema Architecture



```
struct STAFF {
        int staffNo;
        int branchNo;
        char fName [15];
        char lName [15];
        struct date dateOfBirth;
        float salary;
        struct STAFF *next;                    /* pointer to next Staff record */
};
index staffNo; index branchNo;                 /* define indexes for staff */
```

Three Schema Architecture and Data Independence

- The **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
- The **logical level** has a **logical schema**, which describes the structure of the whole database. The logical 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 logical schema
- The **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

Data Independence

- It can be defined as the capacity to change the schema at one level of a database system without having to change the schema at the next higher level.
- We can define two types of data independence:
  - Logical data independence
  - Physical data independence

Logical Data Independence

- It is the capacity to change the logical schema without having to change external schemas or application programs
- We may change the conceptual schema to expand the database (by adding a record type or data item), to change constraints, or to reduce the database (by removing a record type or data item)
- In the last case, only the affected views by the changes will be updated

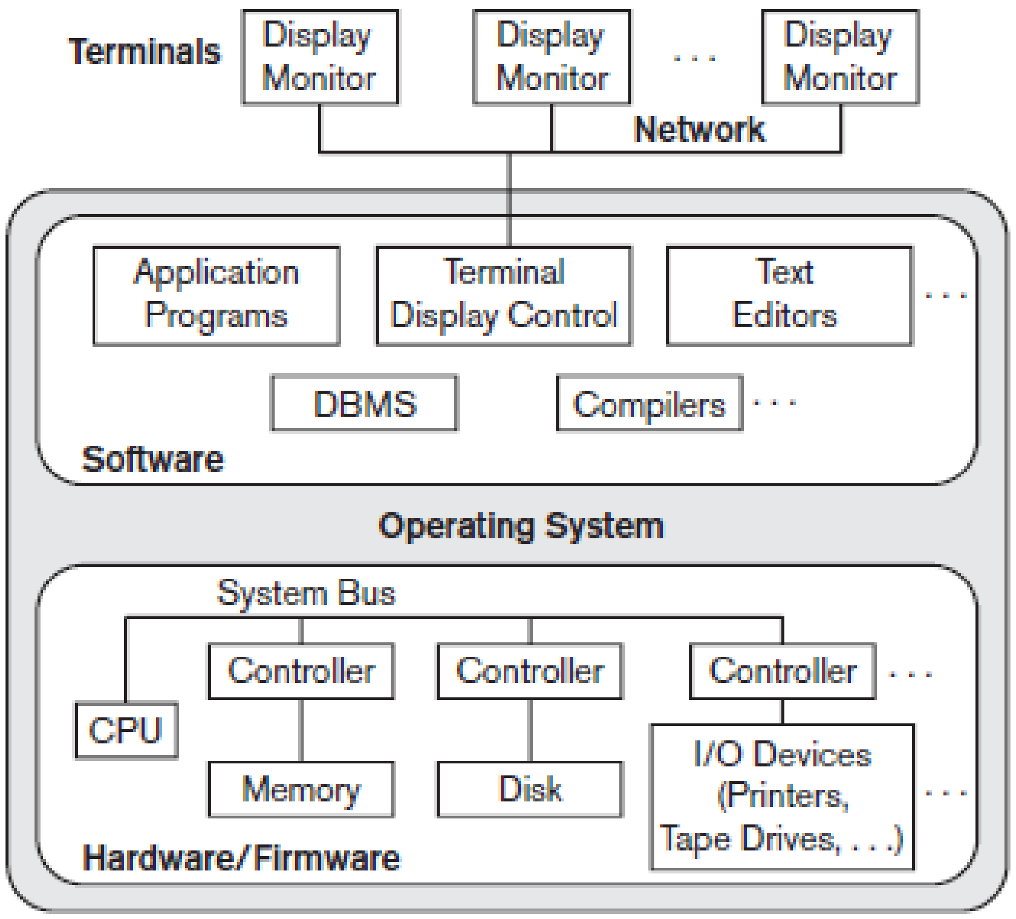
Physical Data Independence

- It is the capacity to change the internal schema without having to change the logical schema. Hence, the external schemas need not be changed as well.
- Changes to the internal schema, such as using different file organizations or storage structures, using different storage devices, modifying indexes or hashing algorithms, should be possible without having to change the conceptual or external schemas. From the users' point of view, the only effect that may be noticed is a change in performance
- In fact, deterioration in performance is the most common reason for internal schema changes

Centralized DBMSs Architecture

- Earlier architectures used mainframe computers to provide the main processing for all system functions, including user application programs and user interface programs, as well as all the DBMS functionality.
- The reason was that most users accessed such systems via computer terminals that did not have processing power and only provided display capabilities.
- Therefore, all processing was performed remotely on the computer system, and only display information and controls were sent from the computer to the display terminals, which were connected to the central computer via various types of communications networks

Centralized DBMSs Architecture



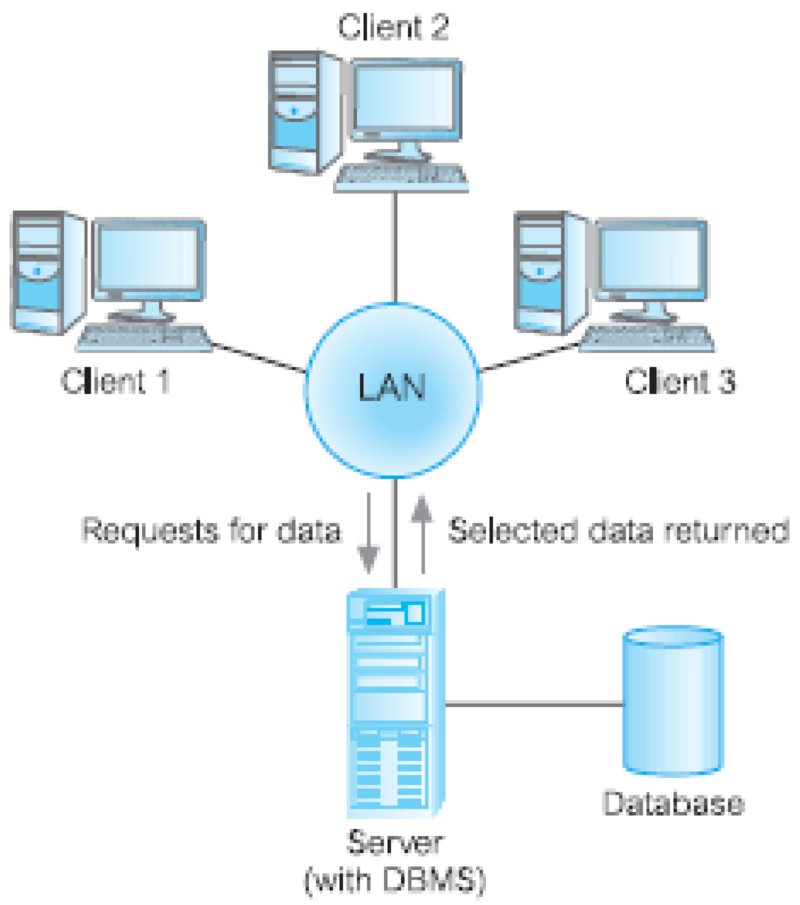
Basic Client/Server Architectures

- The client/server architecture was developed to deal with computing environments in which a large number of PCs, workstations, file servers, print servers, database servers, Web servers, e-mail servers, and other software and equipment are connected via a network.
- The idea is to define specialized servers with specific functionalities
- For example, it is possible to connect a number of PCs or small workstations as clients to a file server that maintains the files of the client machines. Another machine can be designated as a printer server by being connected to various printers; all print requests by the clients are forwarded to this machine
- 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.

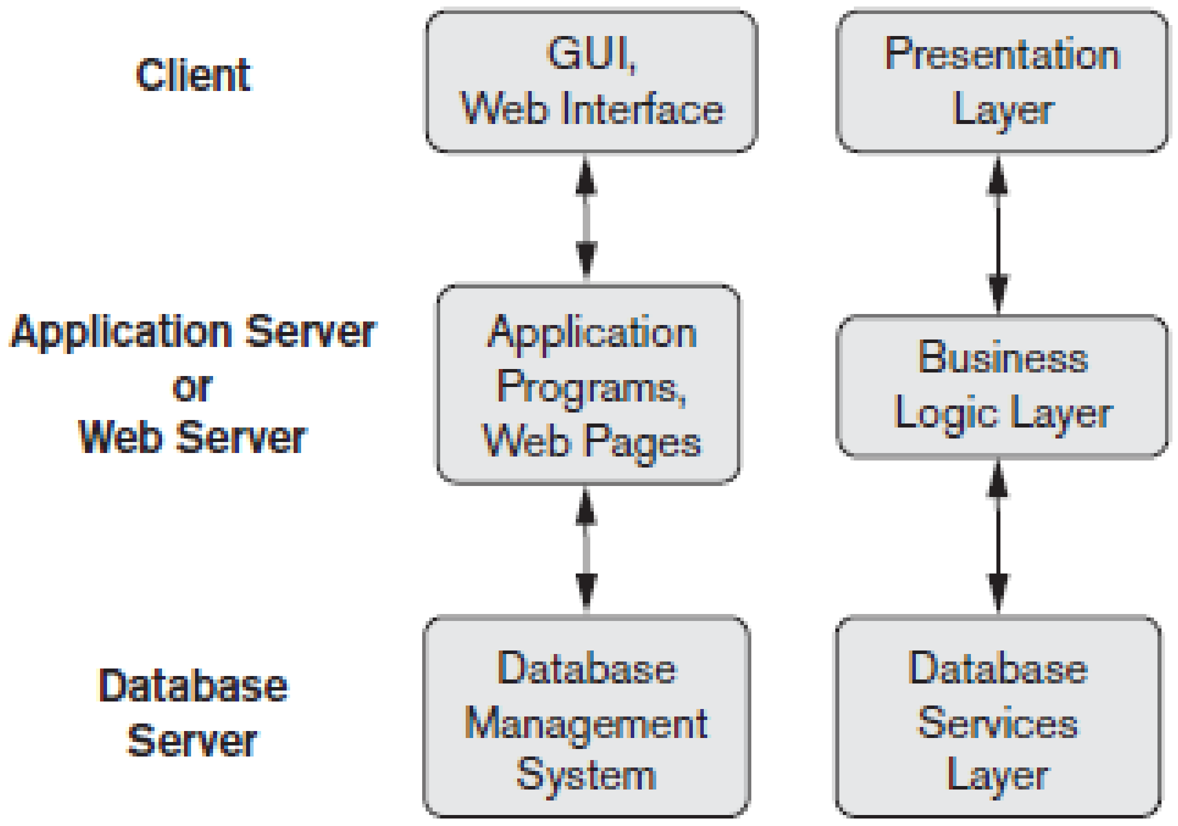
Two-Tier Client/Server Architectures for DBMSs

- This architecture is called two-tier architectures because the software components are distributed over two systems: client and server
- The system components that were first moved to the client side were the user interface and application programs.
- The database, the query and transaction functionality related to SQL processing remained on the server side

Two-Tier Client/Server Architectures for DBMSs



Three-Tier Client/Server Architectures



Three-Tier Client/Server Architectures

- The emergence of the Web changed the roles of clients and servers, leading to the three-tier architecture
- Many Web applications use an architecture called the **three-tier architecture**, which adds an intermediate layer between the client and the database server
- This intermediate layer or **middle tier** is called the **application server** or the **Web server**, depending on the application
- This server plays an intermediary role by running application programs and storing business rules (procedures or constraints) that are used to access data from the database server
- The intermediate server accepts requests from the client, processes the request and sends database queries and commands to the database server, and then returns processed data from the database server to the clients, where it may be processed further and filtered to be presented to users in GUI format.

Three-Tier Client/Server Architectures

- It can improve database security by checking a client's credentials before forwarding a request to the database server.

- It is of great benefit in case of application program changes and updates. These changes need to be applied at only one site not on every workstations where the program is installed

- Thus, the *user interface, application rules,* and *data access* act as the three tiers