1 Introduction

- Flat File Systems
- Database Management System (DBMS)
- Database Applications
- Purpose of DBMS
- Database Users
- DBMS Architecture
- Instances and Schemas
- Data Independence
- Data Models
- Client/Server Architecture

Flat-file Systems

- In early processing systems, an organization's information was stored as groups of records in separate files. The *file processing systems* consisted of a few data files and many application programs. Each file, called a *flat file,* contained processed information for one specific function, such as accounting or inventory.
- A flat file system stores data in a plain text file. Each line of the text file holds one record, with fields separated by delimiters, such as commas or tabs.
- A File Management System (FMS) accommodate flat files that have no relation to other files. This type of database is ideal for a simple databases that do not contain a lot of repeated information. Examples include excel spreadsheet or word data list file.

Database Management System

- A **database** is the collection of related persistent data and contains information relevant to an enterprise. The database is also called the repository or container for a collection of data files. For example, **university database** for maintaining information about *students*, *courses* and *grades* in university.
- A **database system** is basically just a computerized record-keeping system. A database system involves four major components: **data**, **hardware**, **software**, and **users**.
- The **database management system (DBMS)** is the software that handles all access to the database. It is defined as the collection of interrelated data and a set of programs to access those data.

Database Management System

- Users of a DBMS system can perform the following basic operations on the database:
  - Adding new, empty files to the database.
  - Inserting data into existing files.
  - Retrieving data from existing files.
  - Changing data in existing files.
  - Deleting data from existing files.
  - Removing existing files from the database.

Database Applications

- Databases form an essential part of almost all enterprises. Some database applications are given below:
  - **Banking:** For customer information, accounts, and loans, and banking transactions.
  - **Airlines:** For reservation and schedule information.
  - **Universities:** For student information, course registrations, and grades.
  - **Credit card transactions:** For purchase on credit cards and generation of monthly statements.
  - **Telecommunication:** For keeping records of call made, generating monthly bills, maintaining balances on prepaid calling cards, and storing information about the communication networks.
  - **Finance:** For storing information about holdings, sales, and purchase of financial instruments such as stocks and bonds.

Database Applications

- **Sales:** For customer, product, and purchase information.
- **Manufacturing:** For management of supply chain and for tracking production of items in factories, inventories of items in warehouses/stores, and orders for items.
- **Human resources:** For information about employees, salaries, payroll taxes and benefits, and for generation of paychecks.

Purpose of DBMS

- In early days, database applications were built on top of file systems. These systems have many drawbacks. Database systems offer solutions to all the drawbacks of file systems. The benefits using DBMS are:
  - **Redundancy can be reduced:** The database is said to be redundant if the same information is duplicated in several places (data files). For example, the address and telephone number of a particular customer may appear in a file that consists of saving-account records and in a file that consists of checking-account records. We can reduce redundancy by using DBMS.
  - **Inconsistency can be avoided:** The database is said to be inconsistent if various copies of the same data may no longer agree. For example, a changed customer address may be reflected in saving account but not elsewhere in the system. By using DBMS we can avoid inconsistency.

Purpose of DBMS

- **Data can be shared:** The data in the database can be shared among many users and applications.
- **Transaction support can be provided:** A transaction involves several database operations. For example, transfer of a cash amount from account A to account B. In this example two update operations are required.
- **Integrity can be maintained:** The problem of integrity is the problem of ensuring that the data in database is correct. By using DBMS, we can maintain integrity problems.
- **Security can be enforced:** Not every user of the database system should be able to access all data.
- **Efficient Backup and Recovery can be provided:** Provide facilities for recovering from software and hardware failures to reinstate database to previous consistent state.
- **Data in the database can be accessed easily.**

Database Users

- Database users can be classified into two categories: *actors on the scene* and *workers behind the scene.*
- **Actors on the Scene:**
  - These people's jobs involve develop, use, and administer the database. These people are classified into following categories:
  - **Database administrators:** These people are 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. Thus DBA is responsible for the overall control of the system at technical level. Hence, DBA is responsible for the following tasks:
    - Defining Conceptual Schema
    - Defining Internal Schema
    - Defining Security & Integrity Constraints

Database Users

  - Monitoring performance & responsibilities to changing requirements
  - Liaising with users
  - Defining Dump and Reload policies

- **Database Designers:** These people are responsible for defining the content, the structure, the constraints, and functions or transactions against the database. They must communicate with the end-users and understand their needs.
- **End Users:** They use the database for querying, updating and generating reports. End-users can be categorized as follows:
  - **Casual end users:** They access database occasionally when needed. They use sophisticated database query language. They are middle or high-level managers.
  - **Naive or Parametric end users:** 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. For example, bank-tellers, reservation clerks.

Database Users

- **Sophisticated end users:** They have clear knowledge of database system facilities to construct complex queries. They are Engineers, Scientists. They make use of most database facilities.
- **Stand-alone end user:** They make use of personal databases by using ready-made program packages that provide easy-to-use menu based interface.

- **System Analysts and Application Programmers:**
  - **System analysts** determine the requirements of end users, especially naive and parametric end users and develop specifications for canned transactions that meet these requirements.
  - **Application programmers** implement these specifications as programs; then they test, debug, document and maintain these canned transactions.

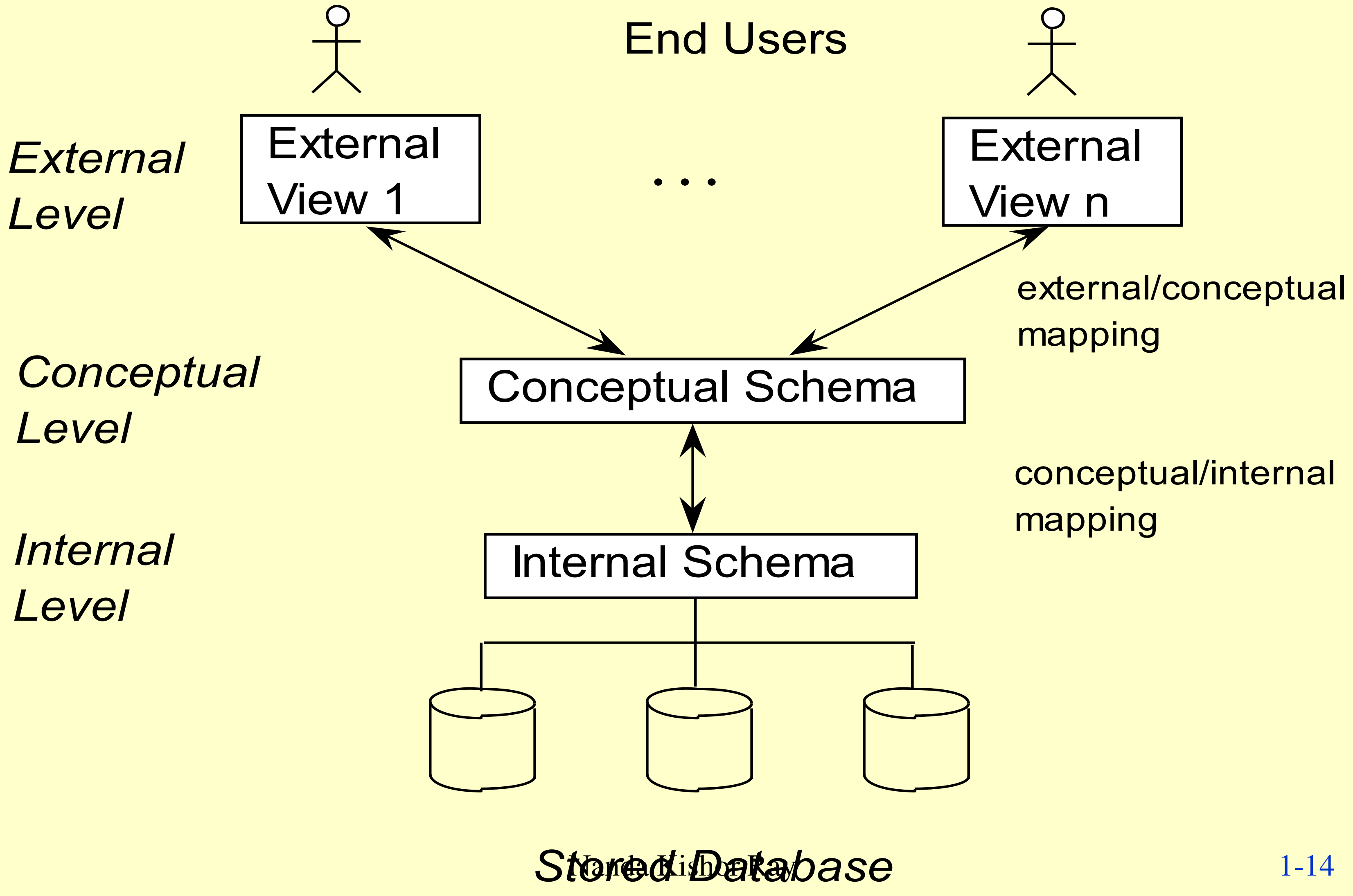
Database Users

- **Workers Behind the Scene:**
  - These people are associated with the design, development, and operation of the *DBMS software and system environment*. These people are not actively interested in the database itself. These people are classified into following types:
  - **DBMS System Designers and implementers:** These people design and implement the DBMS modules and interfaces as a software package.
  - **Tool Developers:** These persons design and implement **tools** – the software packages that facilitate database system design and use and that help improve performance.
  - **Operators and Maintenance Personnel:** These personnel are responsible for the actual running and maintenance of the hardware and software environment for the database system.

DBMS Architecture

- The DBMS architecture proposed by ANSI/SPARC (*American National Standards Institute, Standards Planning And Requirements Committee*) (**ANSI/SPARC architecture**) is defined at three levels. This architecture is also called **three-schema architecture**.
- This architecture provides three levels of abstraction to simplify users' interaction with the system.
- It provides users with an abstract view of data. The system hides certain details of how data are stored and maintained.
- The goal of this architecture is to separate the user applications from physical database.
- It divides the system into three levels of abstraction: the *internal* or *physical level*, the *logical* or *conceptual level*, and the *external* or *view level*.

DBMS Architecture



DBMS Architecture

- **Physical Level or Internal Level:**
  - It is the lowest level of abstraction and describes ***how*** the data in the database are actually stored.
  - This level describes complex low-level data structures in detail and is concerned with the way the data is physically stored.
  - Data only exists at physical level.
- **Logical Level or Conceptual Level:**
  - This is the next higher level of abstraction and describes ***what*** data are stored in the database, and what relationships exist among those data.
  - It describes the structure of whole database and hides details of physical storage structure.
  - It concentrates on describing entities, data types, relationships, attributes and constraints.
  - All of the views must be derivable from this conceptual schema.

DBMS Architecture

- **View Level or External Level:**
  - It is the highest level of abstraction and is concerned with the way the data is seen by individual users.
  - This level simplifies the users’ interaction with the system.
  - It includes a number of user views and hence is guided by the end user requirement.
  - It describes only those part of the database in which the users are interested and hides rest of all from those users. Each user group refers to its own external schema.
- The DBMS must transform a request specified on an **external schema** into a request against the **conceptual schema**, and then into a request on the **internal schema** for processing over the database. The process of transforming requests and results between levels is called **mapping**.

Instances and Schemas

- **Instances:**
  - Databases change over time as information is inserted and deleted. The collection of information stored in the database at a particular moment is called an **instance** of the database.
  - It is also known as **database state**.
- **Schema:**
  - The overall design of the database which is not expected to change frequently is called the database **schema**.
  - There are three schemas, partitioned according to the levels of abstraction. The **physical schema** describes the database design at physical level. The logical **schema describes** the database design at the logical level. The schema at the view level is sometimes called **subschema** and describes the view of the database. A database may have several subschema.

Data Independence

- The three schema architecture further explains the concept of data independence, the capacity to change the schema at one level without having to change the schema at the next higher level.
  - **Logical Data Independence**
  - **Physical 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.

Data Independence

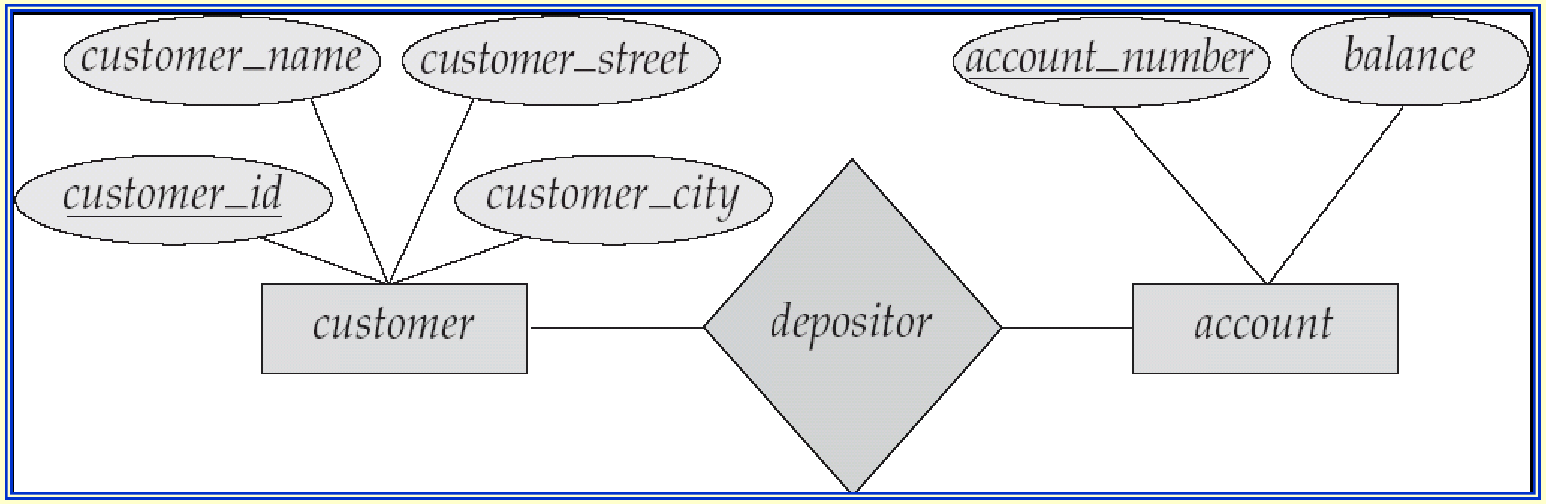
- For example, the internal schema may be changed when certain file structures are reorganized or new indexes are created to improve database performance
- When a schema at a lower level is changed, only the **mappings** between this schema and higher-level schemas need to be changed in a DBMS that fully 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.

Data Models

- The basic structure or design of the database is the **data model**. A data model is a collection of conceptual tools for describing data, data relationships, data semantics, and consistency constraints. Some data models are given below:
- **Entity-Relationship Model:**
  - **Entity-relationship (E-R) model** is a *high level* data model based on a perception of a real world that consists of collection of basic objects, called **entities**, and of **relationships** among these entities.
  - An **entity** is a thing or object in the real world that is distinguishable from other objects.
  - Entities are described in a database by a set of **attributes**.
  - A **relationship** is an association among several entities.
  - The set of all entities of the same type is called an **entity set** and the set of all relationships of the same type is called a **relationship** ***set***.

Data Models

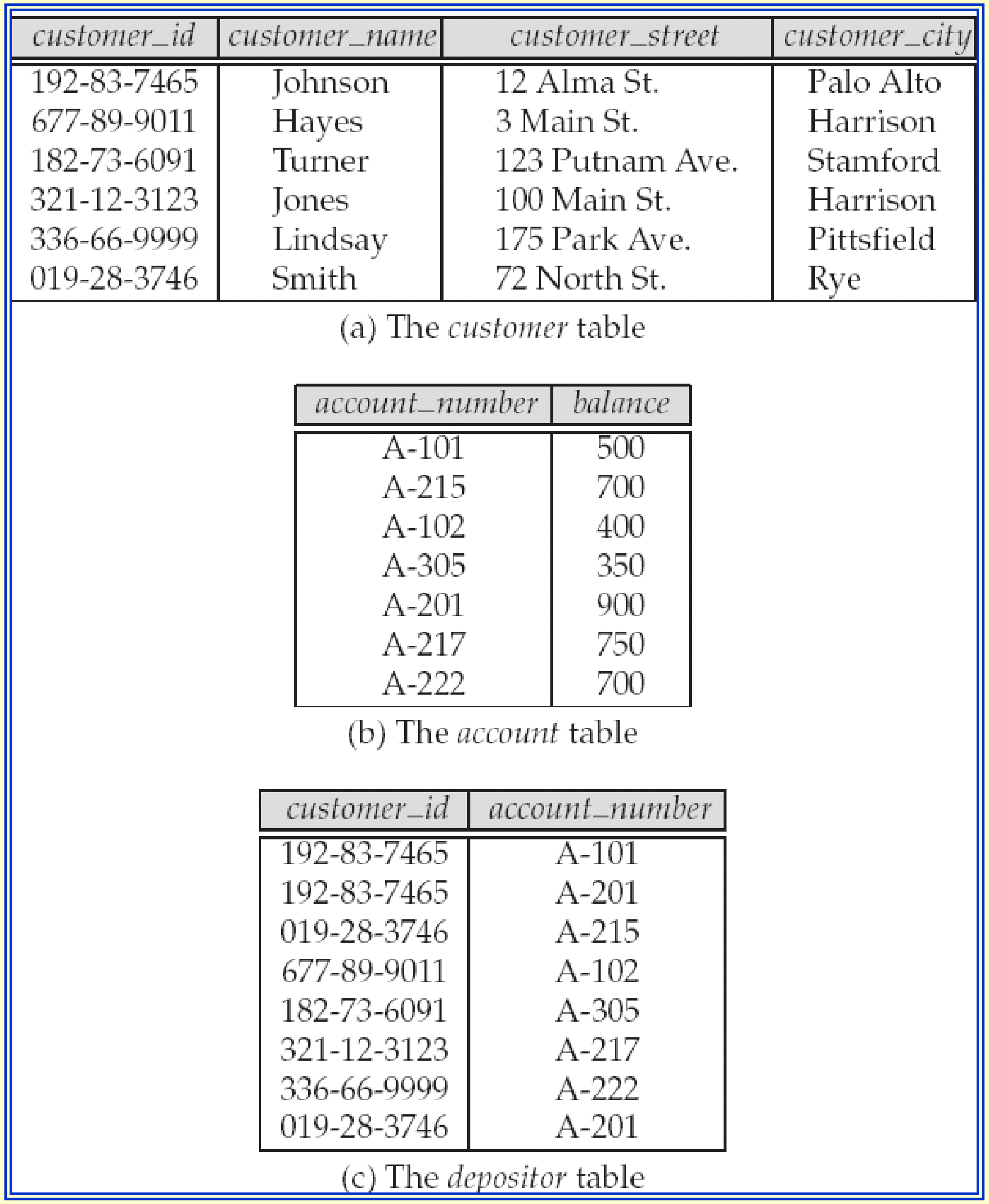
- Overall logical structure of a database can be expressed graphically by E-R diagram. The basic components of this diagram are:
  - **Rectangles** (represent entity sets)
  - **Ellipses** (represent attributes)
  - **Diamonds** (represent relationship sets among entity sets)
  - **Lines** (link attributes to entity sets and entity sets to relationship sets)
- The figure below shows an example of E-R diagram.



Data Models

  - In addition, the E-R model also represents certain constraints to which the contents of the database must conform. The constraints are **mapping cardinalities** and **participation constraints**. (discussed later)

- **Relational Model:**
  - It is the current pervasive model. The relational model is a *lower level* model that uses a collection of tables to represent both data and relationships among those data. Each table has multiple columns, and each column has a unique name. Each table corresponds to an entity set or relationship set, and each row represents an instance of that entity set or relationship set.
  - Relationships link rows from two tables by embedding row identifiers (keys) from one table as attribute values in the other table.
  - Structured query language (SQL) is used to manipulate data stored in tables.



customer_id	*customer_name*	*customer_street*	*customer_city*
192-83-7465	Johnson	12 Alma St.	Palo Alto
677-89-9011	Hayes	3 Main St.	Harrison
182-73-6091	Turner	123 Putnam Ave.	Stamford
321-12-3123	Jones	100 Main St.	Harrison
336-66-9999	Lindsay	175 Park Ave.	Pittsfield
019-28-3746	Smith	72 North St.	Rye

(a) The *customer* table

account_number	*balance*
A-101	500
A-215	700
A-102	400
A-305	350
A-201	900
A-217	750
A-222	700

(b) The *account* table

customer_id	*account_number*
192-83-7465	A-101
192-83-7465	A-201
019-28-3746	A-215
677-89-9011	A-102
182-73-6091	A-305
321-12-3123	A-217
336-66-9999	A-222
019-28-3746	A-201

(c) The *depositor* table

Fig: A sample relational database

Data Models

  - The relational data model is the most widely used data model, and a vast majority of current database systems are based on the relational model. The relational model is at a lower level of abstraction than the E-R model. Database designs are often carried out in the E-R model, and then translated to the relational model.

- **Object-oriented Model:**
  - This model represents an entity set as a class. A class represents both object attributes as well as the behavior of the entity.
  - Instances of class are objects. Within an object, the class attributes takes specific values. However the behavior patterns of the class is shared by all the objects belonging to the class.
  - Attribute values can be primitive data types usually associated with databases and programming languages or other objects. The object-oriented model maintains relationships through ‘logical-containment’.

Data Models

- **Object Relational Model:**
  - This model combines the features of the object-oriented data model and relational data model.
- **Semistructured Model:**
  - This model permits the specification of data where individual data items of the same type may have different set of attributes. The ***extensible markup language*** (XML) is widely used to represent semistructured data.
- **Hierarchical Model:**
  - This model assumes that a tree structure is the most frequently occurring relationship.
- **Network Model:**
  - The network model replaces the hierarchical tree with a graph thus allowing more general connections among the nodes. This model was evolved to specially handle non-hierarchical relationships.

Client/Server Architecture

- In client/server architecture, *user interface* and *application programs* are located on the client side and *query* and *transaction facility* are located on the server side.
- The server is often called a *query server* or *transaction server* because it provides these two functionality. It is also called an *SQL server* since most servers are based on the SQL language and standard.
- The user interface programs and application programs can run on the client side.
- When DBMS access is required, the program establishes a connection to the DBMS which is on the server side.
- Once the connection is created, the client program can communicate with the DBMS.
- Some interfaces that provide an *application programming interface (API)* that allow client side programs to call the DBMS are *ODBC (Open Database Connectivity)* and *JDBC (Java Database Connectivity).*

Client/Server Architecture

- Other variations of clients are also possible. For example, in some DBMSs, more functionality is transferred to clients including *user interface, data dictionary functions*, *interactions with programming language compilers, global query optimization, concurrency control,* and *recovery across multiple servers*. In such situations the server may be called the *Data Server.*
- The architecture described so far is called **two-tier architecture** because the software components are distributed over two systems: client and server.
- Many Web applications use an architecture called the **three-tier architecture** which adds an intermediate layer (**middle tier** or **application server** or **Web server**) between the client and the database server.
- This server plays an intermediary role by storing business rules (procedures or constraints) that are used to access data from the database server.

Client/Server Architecture

- It can also improve database security by checking a client's credentials before forwarding a request to the database server.
- Clients contain GUI interfaces and some additional application-specific business rules.
- Advances in encryption and decryption technology make it safer to transfer sensitive data from server to client in encrypted form, where it will be decrypted.