MODULE I

DATA: STRUCTURED, SEMI-STRUCTURED AND UNSTRUCTURED DATA

- **Data** is a collection of facts and figures that can be processed to produce information.
- **Database** is a collection of related data, Database is one of the important components for many applications and is used for storing a series of data in a single set. In other words, it is a group / package of information that is put in order so that it can be easily access, manage and update. There are different types of database.
  They are:
  Bibliographic
  fulltext
  numeric
  images

In a database, even a smallest portion of information becomes the data. Example, Student is a data, roll number is a data, and address is a data, height, weight, marks everything is data. In brief, all the living and nonliving objects in this world is a data

- A **database management system** stores data in such a way that it becomes easier to retrieve, manipulate, and produce information. A **database-management system (DBMS)** is a collection of interrelated data and a set of programs to access those data. The collection of data, usually referred to as the **database**, contains information relevant to an enterprise. The primary goal of a DBMS is to provide a way to store and retrieve database information that is both *convenient* and *efficient*.

- **DBMS**
  A DBMS allows creation, definition and manipulation of database. DBMS is actually a tool used to perform any kind of operation on data in database. Dbms also provides protection and security to database. It maintains data consistency in case of multiple users. Here are some examples of popular dbms, MySql, Oracle, Sybase, Microsoft Access and IBM DB2 etc.

DATABASE-SYSTEM APPLICATIONS

- *Enterprise Information*

◦ ***Sales***: For customer, product, and purchase information.
◦ ***Accounting***: For payments, receipts, account balances, assets and other accounting information.
◦ ***Human resources***: For information about employees, salaries, payroll taxes, and benefits, and for generation of paychecks.
◦ ***Manufacturing***: For management of the supply chain and for tracking production of items in factories, inventories of items inwarehouses and stores, and orders for items.
◦ ***Online retailers***: For sales data noted above plus online order tracking, generation of recommendation lists, and maintenance of online product evaluations.

- *Banking and Finance*

◦ ***Banking***: For customer information, accounts, loans, and banking transactions.
◦ ***Credit card transactions***: For purchases on credit cards and generation of monthly statements.
◦ ***Finance***: For storing information about holdings, sales, and purchases of financial instruments such as stocks and bonds; also for storing real-time market data to enable online trading by customers and automated trading by the firm.

- *Universities*: For student information, course registrations, and grades (in addition to standard enterprise information such as human resources and accounting).

- *Airlines*: For reservations and schedule information. Airlines were among the first to use databases in a geographically distributed manner.

- *Telecommunication*: For keeping records of calls made, generating monthly bills, maintaining balances on prepaid calling cards, and storing information about the communication networks.

Structured data
The information stored in databases is known as structured data because it is represented in a strict format.

For example, each record/tuple in a relational database table-such as the EMPLOYEE table in Figure below follows the same format as the other records in that table

.

The DBMS then checks to ensure that all data follows the structures and constraints specified in the schema.

Semi-structured data

In some applications, data is collected in an ad-hoc manner before it is known how it will be stored and managed.

This data may have a certain structure, but not all the information collected will have identical structure. Some attributes may be shared among the various entities, but other attributes may exist only in a few entities.

Moreover, additional attributes can be introduced in some of the newer data items at any time, and there is no predefined schema. This type of data is known as semi-structured data.

A key difference between structured and semi-structured data concerns how the schema constructs (such as the names of attributes, relationships, and entity types) are handled.

In semi - structured data, the schema information is mixed in with the data values, since each data object can have different attributes that are not known in advance. Hence, this type of data is sometimes referred to as **self-describing data.**

Tree or graph data structure represent semi-structured data.

Semi-structured data may be displayed as a directed graph, as shown in Figure below. The labels or tags on the directed edges represent the schema names: the names of attributes, object types (or entity types or classes), and relationships. The internal nodes represent individual objects or composite attributes. The leaf nodes represent actual data values of simple (atomic) attributes.

There are two main differences between the semi-structured model and the object model :

1. The schema information-names of attributes, relationships, and classes (object types) in the semi-structured model is intermixed with the objects and their data values in the same data structure.

2. In the semi-structured model, there is no requirement for a predefined schema to which the data objects must conform.

Unstructured data

A typical example is a text document that contains information embedded within it. Web pages in HTML that contain some data are considered to be unstructured data.

Hence, the tags (Eg:<html>,<table><font>) in HTML document specify document formatting rather than the meaning of the various data elements in the document.

HTML tags specify information, such as font size and style (boldface, italics, and so on), color, heading levels in documents, and so on. Some tags provide text structuring in documents, such as specifying a numbered or unnumbered list or a table.

Even these structuring tags specify that the embedded textual data is to be displayed in a certain manner, rather than indicating the type of data represented in the table.

STRUCTURED DATA	**SEMI STRUCTURED DATA**	**UNSTRUCTURED DATA**
• Strict format • All records have same format • Structured data is easily organized and generally stored in databases. • Structured data generally consists of numerical information and is objective • ATM, AIRLINE,SALES • Readily searchable, high degree of organization	• Data may have certain structure but not all information collected has same structure. • Semistructured data is data that is neither raw data, nor typed data in a conventional database system. It is structured data, but it is not organized in a rational model, like a table or an object based graph. • A lot of data found on the Web can be described as semistructured. • Data integration especially makes use of semistructured data.	• No predetermined form or structure • Very limited indication of data type • Compilation is a time and energy consuming task • Unstructured data can also come from social media sites such as Facebook, LinkedIn, Twitter, Tumblr, Flickr, Yelp, YouTube, and Pinterest. • Some examples of unstructured data include customer reviews that describe how they feel about an experience, identified triggers for readmission to hospital care, or call center conversations.

CONCEPT & OVERVIEW OF DBMS

Database: A collection of related data.

Data: Known facts that can be recorded and have an implicit meaning.

Mini-world: Some part of the real world about which data is stored in a database. For example, student grades and transcripts at a university.

Database Management System (DBMS): A collection of programs/general-purpose software to facilitate users to create and maintain a database.(defining, constructing, manipulating, sharing databases among various users.)

Database System: The DBMS software together with the database .Sometimes, the applications are also included.

Data Models

(1)High-level or conceptual data models provide concepts that are close to the way many users **perceive data**. Conceptual data models use concepts such as entities, attributes, and relationships. **Object data models** as a new family of higher-level implementation data models.

(a)Entity-Relationship Model. The entity-relationship (E-R) data model uses a collection of basic objects,called entities,and relationships among these objects. An entity is a "thing" or "object" in the real world that is distinguishable from other objects.

(b)Object-BasedDataModel Object-oriented programming(especiallyinJava, C++, or C#) has become the dominant software-development methodology. This led to the development of an object-oriented data model that can be seen as extending the E-R model with notions of encapsulation, methods (functions), and object identity.

(2)Low-level or physical data models provide concepts that describe the details of **how data is stored** in the computer. Concepts provided by low-level data models are generally meant for computer specialists, not for typical end users. Physical data models describe how data is stored **as files in** the computer by representing information such as record formats, record orderings, and **access paths**. An **access path** is a structure that makes the **search** for particular database records **efficient.**

(3)Between these two extremes is a class of **representational (or implementation) data models**, which provide concepts that may be understood by end users but that are not too far removed from the way data is organized within the computer. Representational data models **hide some details of data storage** but can be implemented on a computer system in a direct way.

Representational or implementation data models **include** the widely used **relational data model**, as well as the so-called legacy data models-**the network and hierarchical model.**

(a)Relational Model. The relational model uses a collection of tables to represent both data and the relationships among those data. Each table has multiple columns, and each column has a unique name. Tables are also known as relations.

Database Administrator

The database administrator (DBA) is the central authority for managing a database system.

The DBA's responsibilities include granting privileges to users who need to use the system and classifying users and data in accordance with the policy of the organization.

Database administrators are responsible for authorizing access to the database, for co-ordinating and monitoring its use, acquiring software, and hardware resources, controlling its use and monitoring efficiency of operations.

1. *Account creation*

2. *Privilege granting*

3. *Privilege revocation(cancel)*

4. *Security level assignment*

1.**Account creation:** This action creates a new account and password for a user or a group of users to enable access to the DBMS.

2.**Privilege granting:** This action permits the DBA to grant certain privileges to certain accounts.

3.**Privilege revocation:** This action permits the DBA to revoke (cancel) certain privileges that were previously given to certain accounts.

4.**Security level assignment:** This action consists of assigning user accounts to the appropriate security classification level.

The DBA is responsible for the overall security of the database system.

Action 1 in the preceding list is used to control access to the DBMS as a whole, whereas **Actions 2 and 3** are used to control discretionary database authorization, and **Action 4** is used to control mandatory authorization.

Database Users

• **Naive users :** are unsophisticated users who interact with the system by invoking one of the application programs that have been written previously. For example, a clerk in the university who needs to add a new instructor to department A invokes a program called new hire. This program asks the clerk for the name of the new instructor, her new ID, the name of the department (that is, A), and the salary. The typical user interface for naive users is a forms interface, where the user can fill in appropriate fields of the form. Naive users may also simply read reports generated from the database.

As another example, consider a student, who during class registration period, wishes to register for a class by using a Web interface. Such a user connects to a Web application program that runs at a Web server.

•**Application programmers** are computer professionals who write application programs. Application programmers can choose from many tools to develop user interfaces.

• **Sophisticated users** interact with the system without writing programs. Instead, they form their requests either using a database query language or by using tools such as data analysis software. Analysts who submit queries to explore data in the database fall in this category.

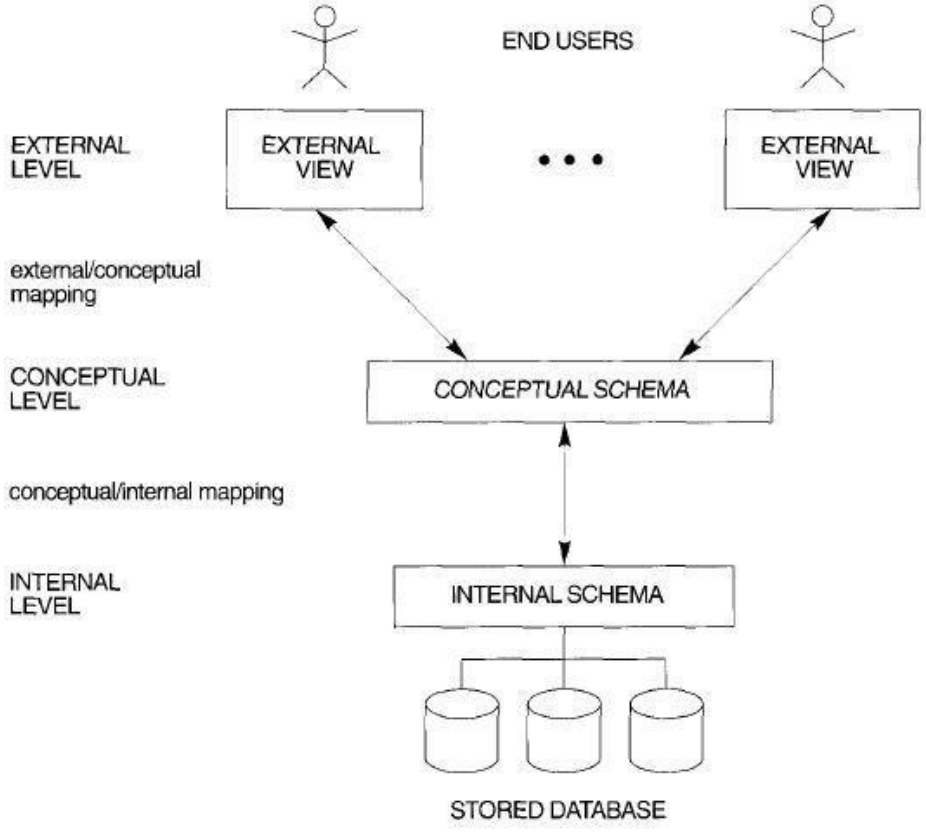
• **Specialized users** are sophisticated users who write specialized database applications that do not fit into the traditional data-processing framework.

THREE SCHEMA ARCHITECTURE OF DBMS.

The goal of the three-schema architecture, illustrated in Figure below, is to separate the user applications and the physical database

1. 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.

2. The conceptual level has a conceptual schema, which describes the structure of the whole database for a community of users. The conceptual schema hides the details of physical storage



The three-schema architecture.

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 when a database system is implemented. This implementation conceptual schema is often based on a conceptual schema design in a high-level data model.

3. 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. As in the previous case, each external schema is typically implemented using a representational data model, possibly based on an external schema design in a high level data model.

- In a DBMS based on the three-schema architecture, each user group refers only to its own external schema.

- Hence, 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 stored database.

- If the request is a database retrieval, the data extracted from the stored database must be reformatted to match the user's external view.

- The processes of transforming requests and results between levels are called mappings.

DATA INDEPENDENCE

Upper levels are not affected by change in lower level. The three-schema architecture can be used to further explain the concept of data independence, which 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:

PHYSICAL DATA INDEPENDENCE	LOGICAL DATA INDEPENDENCE
• Physical storage or devices can be changed without affecting conceptual schema • Modification at this level is to improve performance • Not difficult (just changing the location of data)	• Conceptual schema can be changed without affecting the external schema • Structure of database is altered in this level • Difficult (depends on the affecters)

1. **Logical data independence** is the capacity to change the conceptual 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, external schemas that refer only to the remaining data should not be affected.

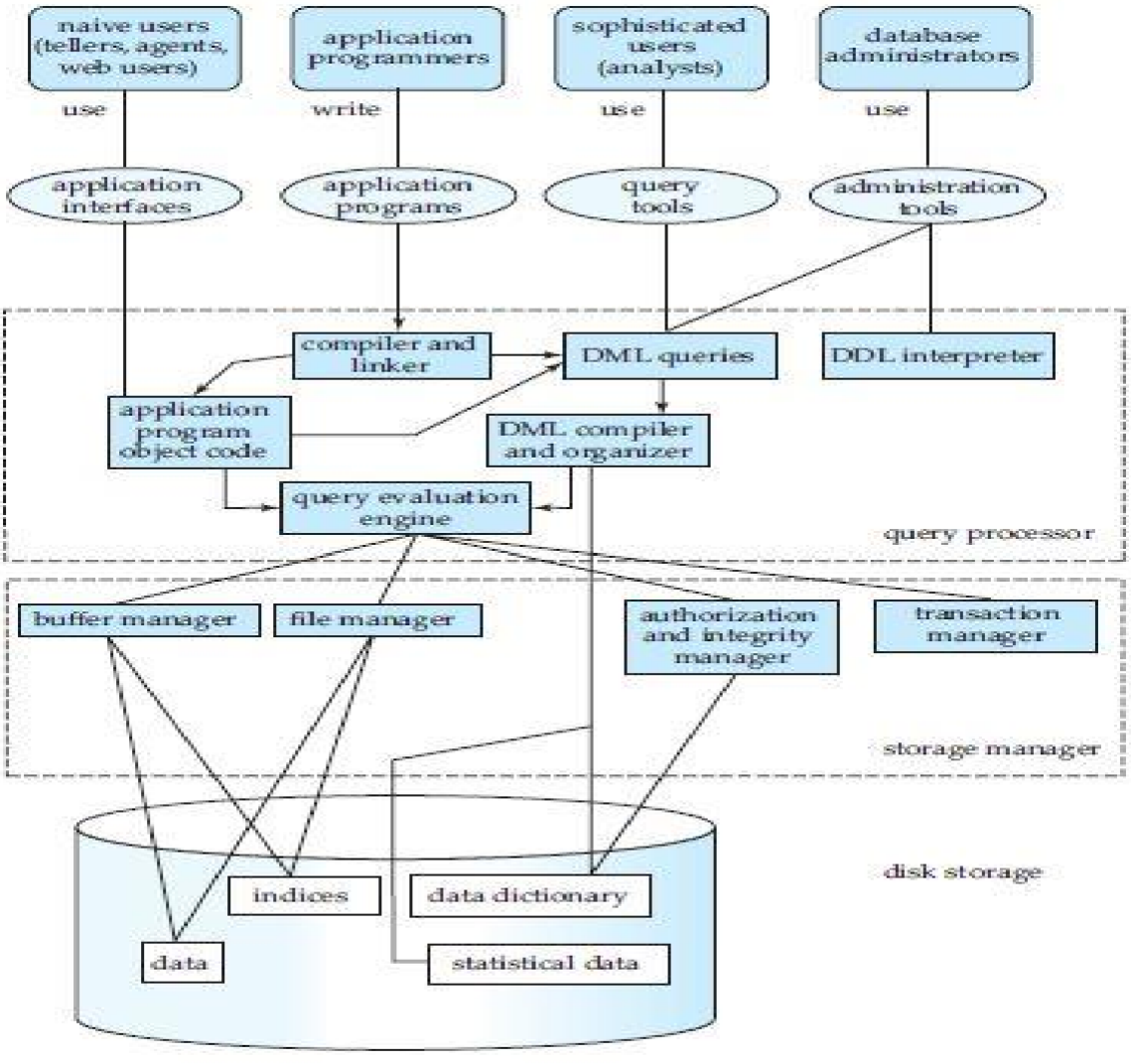
2. **Physical data independence** is the capacity to change the internal schema without having to change the conceptual schema. Hence, the external schemas need not be changed as well. Changes to the internal schema may be needed because some physical files had to be reorganized-for example, by creating additional access structures-to improve the performance of retrieval or update. If the same data as before remains in the database, we should not have to change the conceptual schema.

DATABASE ARCHITECTURES AND CLASSIFICATION

The architecture of a database system is greatly influenced by the underlying computer system on which the database system runs. Database systems can be centralized, or client-server, where one server machine executes work on behalf of multiple client machines. Database systems can also be designed to exploit parallel computer architectures. Distributed databases span multiple geographically separated machines.

(a)Centralized DBMSS Architecture

Earlier architectures used mainframe computers to provide the main processing for all functions of the system, including user application programs and user interface programs, as well as all the DBMS functionality.

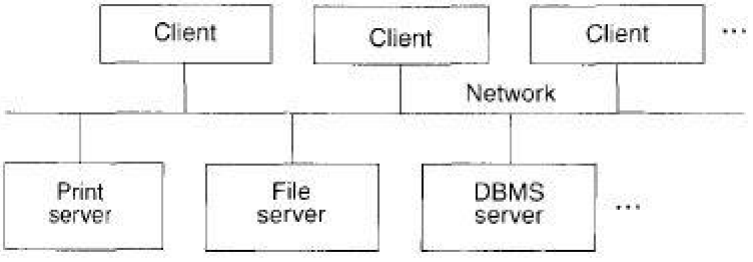


System structure.

The reason was that most users accessed such systems via computer terminals that did not have processing power and only provided display capabilities. So, 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.

(b)Two-Tier Client/Server Architectures for DBMS

In such a client/server architecture, 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.

A standard called Open Database Connectivity (ODBC) provides an application programming interface (API), which allows client-side programs to call the DBMS, as long as both client and server machines have the necessary software installed. Most DBMS vendors provide ODBC drivers for their systems.

Hence, a client program can actually connect to several RDBMSs and send query and transaction requests using the ODBC API, which are then processed at the server sites. Any query results are sent back to the client program, which can process or display the results as needed.

The architectures described here are called two-tier architectures because the software components are distributed over two systems: client and server.

The advantages of this architecture are its simplicity and seamless compatibility with existing systems. The emergence of the World Wide Web changed the roles of clients and server, leading to the three-tier architecture.

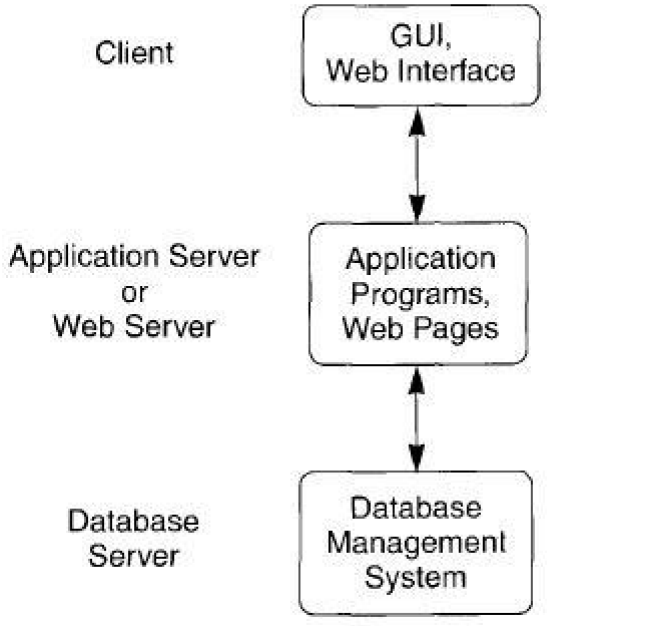
(c)Three-Tier Client/Server Architectures for Web Applications

Many Web applications use an architecture called the three-tier architecture, which adds an intermediate layer between the client and the database server, as illustrated in Figure 2.7. This intermediate layer or middle tier is sometimes called the application server and sometimes the Web server, depending on the application.

This server plays an intermediary role by storing business rules (procedures or constraints) that are used to access data from the database server. 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.

The intermediate server accepts requests from the client, processes the request and sends database commands to the database server, and then acts as a conduit for passing (partially) processed data from the database server to the clients, where it may be processed further

filtered to be presented to users in GUI format. Thus, the user interface, application rules, and data access act as the three tiers.



ENTITY-RELATIONSHIP MODEL: BASIC CONCEPTS

- The E-R data model employs three ***basic concepts:***

  *entity sets

  *relationship sets and

  *attributes

1. Entity Sets

An entity is a “thing” or “object” in the real world that is distinguishable from all other objects. For example, each person in a university is an entity. An entity has a set of properties, and the values for some set of properties may uniquely identify an entity. For instance, a person may have a person id property whose value uniquely identifies that person.

An entity set is a set of entities of the same type that share the same properties, or attributes. The set of all people who are instructors at a given university, for example, can be defined as the entity set instructor. Similarly , the entity set student might represent the set of all students in the university.

An entity is represented by a set of attributes. Attributes are descriptive properties possessed by each member of an entity set. Possible attributes of the instructor entity set are ID, name, dept name, and salary. Each entity has a value for each of its attributes. For instance, a particular instructor entity may have the value 12121 for ID.

2. Relationship Sets

A relationship is an association among several entities. For example, we can define a relationship advisor that associates instructor Katz with student Shankar.

A relationship set is a set of relationships of the same type.

Consider the two entity sets instructor and student in Figure 7.1. We define the relationship set advisor to denote the association between instructors and students. Figure 7.2 depicts this association

The association between entity sets is referred to as participation; that is, the entity sets E1, E2,...,En participate in relationship set R.

The function that an entity plays in a relationship is called that entity's role.

A relationship may also have attributes called descriptive attributes. Consider a relationship set advisor with entity sets instructor and student. We could associate the attribute date with that relationship to specify the date when an instructor became the advisor of a student.

The relationship sets advisor and dept advisor provide examples of a binary relationship set—that is,one that involves two entity sets. The number of entity sets that participate in a relationship set is the degree of the relationship set.

(3)Attributes

For each attribute, there is a set of permitted values, called the domain, or value set, of that attribute. The domain of attribute course id might be the set of all text strings of a certain length.

An attribute, as used in the E-R model, can be characterized by the following attribute types:

(a)Simple and composite attributes

The attributes that are simple; that is, they have not been divided into subparts. Composite attributes, on the other hand, can be divided into subparts (that is, other attributes).

For example,an attribute name could be structured as a composite attribute consisting of first name, middle _initial,and last name. The address can be defined as the composite attribute address with the attributes street, city, state, and zip code.

(b)Single-valued and multivalued attributes

For instance,the student ID attribute for a specific student entity refers to only one student ID. Such attributes are said to be single valued. There may be instances where an attribute has a set of values for a specific entity.

An instructor may have zero, one, or several phone numbers, and different instructors may have different numbers of phones. This type of attribute is said to be multivalued.

(d)Derived attribute

The value for this type of attribute can be derived from the values of other related attributes or entities. Suppose that the instructor entity set has an attribute age that indicates the instructor's age.

If the instructor entity set also has an attribute date of birth, we can calculate age from date of birth and the current date.Thus ,age is a derived attribute. In this case, date of birth may be referred to as a base attribute, or a stored attribute. The value of a derived attribute is not stored but is computed when required.

An attribute takes a null value when an entity does not have a value for it. The null value may indicate "not applicable"—that is, that the value does not exist for the entity. For example, one may have no middle name.

Null can also designate that an attribute value is unknown. An unknown value may be either missing (the value does exist, but we do not have that information) or not known.

DESIGN ISSUES

1.Use of Entity set vs. Attributes.

In the real world situations, sometimes it is difficult to select the property as an attribute or an entity set

2.Use of Entity Sets versus Relationship Sets

Sometimes, an entity set can be better expressed in relationship set. Thus, it is not always clear whether an object is best expressed by an entity set or a relationship set

3.Binary versus n-ary Relationship Sets

Relationships in databases are often binary. Some relationships that appear to be non-binary could actually be better represented by several binary relationships

4.Placement of Relationship Attributes

The cardinality ratio of a relationship can affect the placement of relationship attributes:

- **One-to-Many**: Attributes of 1:N relationship set can be repositioned to **only** the entity set on the many side of the relationship.

- **One-to-One**: The relationship attribute can be associated with **either one** of the participating entities

- **Many-to-Many**: Here, the relationship attributes can not be represented to the entity sets; rather they will be represented as attribute of relationship set.

MAPPING CONSTRAINTS

An E-R enterprise schema may define certain constraints to which the contents of a database must conform.

For a binary relationship set R between entity sets A and B, the mapping cardinality must be one of the following:

• One-to-one. An entity in A is associated with at most one entity in B, and an entity in B is associated with at most one entity in A. (See Figure 7.5a.)

• One-to-many. An entity in A is associated with any number (zero or more) of entities in B. An entity in B, however, can be associated with at most one entity in A. (See Figure 7.5b.)

• Many-to-one. An entity in A is associated with at most one entity in B. An entity in B, however, can be associated with any number (zero or more) of entities in A. (See Figure 7.6a.)

• Many-to-many.An entity in A is associated with any number(zero or more) of entities in B, and an entity in B is associated with any number (zero or more) of entities in A. (See Figure 7.6b.)

KEYS

Super Key: The set of attributes which can uniquely identify a tuple is known as Super Key. For Example, STUD_NO, (STUD_NO, STUD_NAME) etc.

A candidate key is a super key but vice versa is not true

Candidate Key: The minimal set of attribute which can uniquely identify a tuple is known as candidate key. For Example, STUD_NO in STUDENT relation.

There can be more than one candidate key in a relation. For Example, STUD_NO as well as STUD_PHONE both are candidate keys for relation STUDENT.

The candidate key can be simple (having only one attribute) or composite as well. For Example, {STUD_NO, COURSE_NO} is a composite candidate key for relation STUDENT_COURSE.

Primary Key: There can be more than one candidate key in a relation out of which one can be chosen as primary key. For Example, STUD_NO as well as STUD_PHONE both are candidate keys for relation STUDENT but STUD_NO can be chosen as primary key (only one out of many candidate keys).

Alternate Key: The candidate key other than primary key is called as alternate key. For Example, STUD_NO as well as STUD_PHONE both are candidate keys for relation STUDENT but STUD_PHONE will be alternate key (only one out of many candidate keys).

ENTITY- RELATIONSHIP DIAGRAM

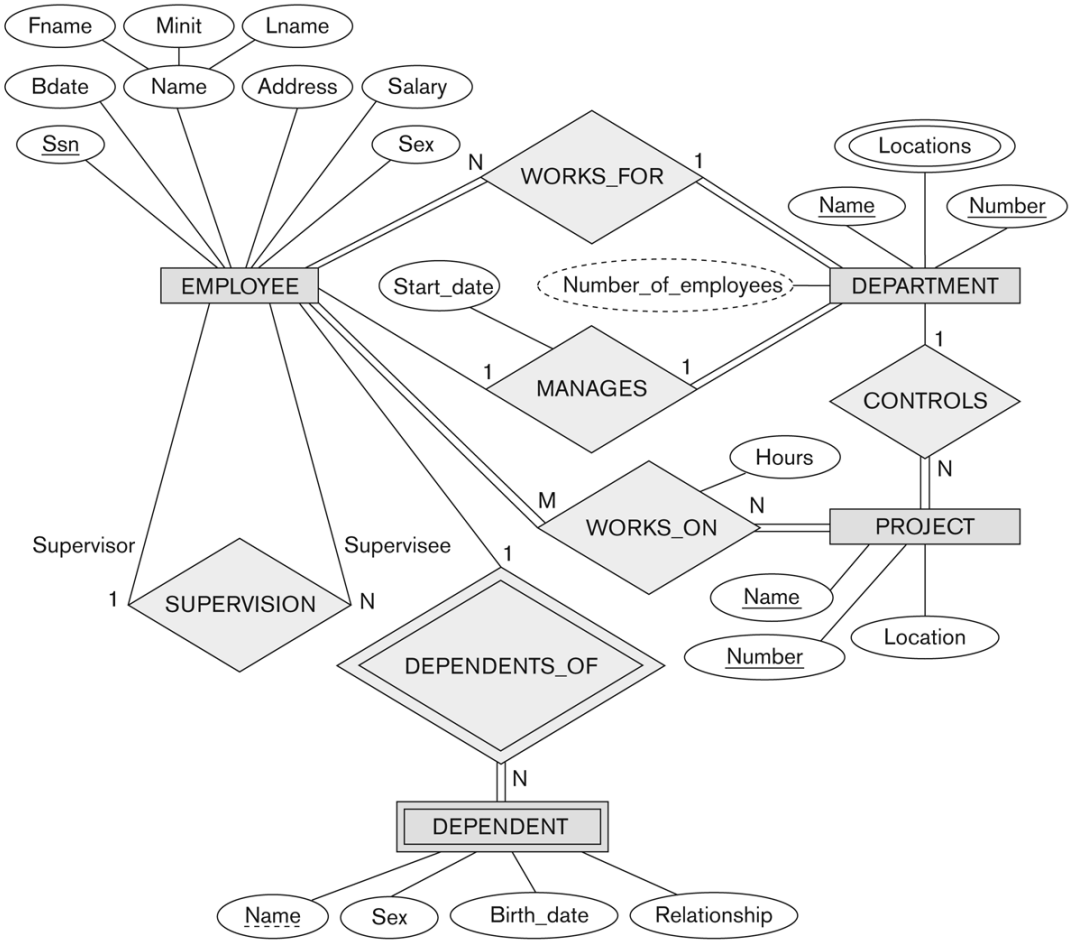


Figure 3.2
An ER schema diagram for the COMPANY database. The diagrammatic notation is introduced gradually throughout this chapter.

We describe an example database application, called COMPANY, that serves to illustrate the basic ER model concepts.

The COMPANY database keeps track of a company's employees, departments, and projects. Suppose that after the requirements collection and analysis phase, the database designers provided the following description of the "miniworld"-the part of the company to be represented in the database:

1. The company is organized into departments. Each department has a unique name, a unique number, and a particular employee who manages the department. We keep track of the start date when that employee began managing the department. A department may have several locations.
2. A department controls a number of projects, each of which has a unique name, a unique number, and a single location.
3. We store each employee's name, social security number.i address, salary, sex, and birth date. An employee is assigned to one department but may work on several projects, which are not necessarily controlled by the same department. We keep track of the number of hours per week that an employee works on each project. We also keep track of the direct supervisor of each employee.
4. We want to keep track of the dependents of each employee for insurance purposes. We keep each dependent's first name, sex, birth date, and relationship to the employee.

Figure 3.2 shows how the schema for this database application can be displayed by means of the graphical notation known as ER diagrams

WEAK ENTITY SETS

- Entity types that do not have key attributes(primary key) of their own are called weak entity types.
- In contrast, regular entity types that do have a key attribute are called strong entity types.

- Entities belonging to a weak entity type are identified by being related to specific entities from another entity type in combination with one of their attribute values.

- We call this other entity type the identifying or owner entity type, and we call the relationship type that relates a weak entity type to its owner the identifying relationship of the weak entity type.

- A weak entity type always has a total participation constraint (existence dependency) with respect to its identifying relationship, because a weak entity cannot be identified without an owner entity.

- However, not every existence dependency results in a weak entity type.

For example, a DRIVER_LICENSE entity cannot exist unless it is related to a PERSON entity, even though it has its own key (LicenseNumber) and hence is not a weak entity.

Consider the entity type DEPENDENT, related to EMPLOYEE, which is used to keep track of the dependents of each employee via a l:N relationship .

The attributes of DEPENDENT are Name (the first name of the dependent), BirthDate, Sex, and Relationship (to the employee). Two dependents of two distinct employees may, by chance, have the same values for Name, BirthDate, Sex, and Relationship, but they are still distinct entities. They are identified as distinct entities only after determining the particular employee entity to which each dependent is related. Each employee entity is said to own the dependent entities that are related to it.

- A weak entity type normally has a partial key, which is the set of attributes that can uniquely identify weak entities that are related to the same owner entity.

In our example, if we assume that no two dependents of the same employee ever have the same first name, the attribute Name of DEPENDENT is the partial key. In the worst case, a composite attribute of all the weak entity's attributes will be the partial key.

- In ER diagrams, both a weak entity type and its identifying relationship are distinguished by surrounding their boxes and diamonds with double lines. The partial key attribute is underlined with a dashed or dotted line.