**MODULE I**

**Q) Explain Database and Database Management system**

**Q) List the various functions of DBMS**

A database is a collection of correlated data. By data we mean known facts that can be recorded and that have implicit meaning. For example, consider the names, telephone, numbers, and addresses of the people you know.

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.

A database management system (DBMS) is a collection of programs that enables users to create and maintain a database.

.

**Functions of DBMS**

The DBMS is a *general-purpose software system* that facilitates the processes of *defining, constructing, manipulating,* and *sharing* databases among various users and applications.

* **Defining** a database involves specifying the data types, structures, and constraints of the data to be stored in the database. The database definition or descriptive information is also stored by the DBMS in the form of a database catalog or dictionary; it is called **meta-data.**
* **Constructing** the database is the process of storing the data on some storage medium that is controlled by the DBMS.
* **Manipulating** a database includes functions such as querying the database to retrieve specific data, updating the database to reflect changes in the miniworld, and generating reports from the data.
* **Sharing** a database allows multiple users and programs to access the database simultaneously.
* An **application program** accesses the database by sending queries or requests for data to the DBMS. A **query** typically causes some data to be retrieved.

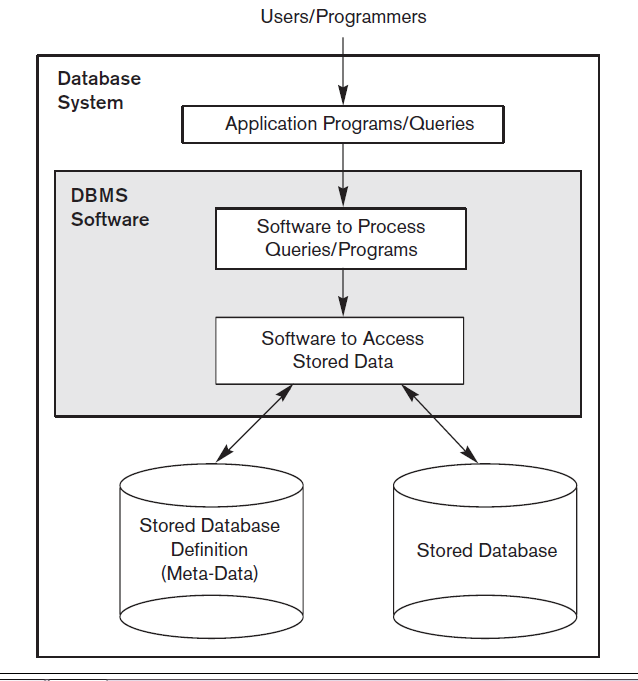
Other important functions provided by the DBMS include

- **protecting** the database and **maintaining** it over a long period of time. Protectionincludes system protection against hardware or software malfunction (or crashes) and security protection against unauthorized or malicious access.

**We call the database and DBMS together database system**

* A simplified database system environment

.



**Q) Explain the advantages and disadvantages of Database approach over File Processing approach**(can write the 4 points below as well as points under Advantages of Database for both the questions)

**Q) Features/Characteristics of Database approach**

**Database Approach v/s File Processing approach**

In traditional **file processing**, each user defines and implements the files needed for a specific software application as part of programming the application.

For example, one user, the *grade reporting office,* may keep files on students and their grades.. A second user, the *accounting office*, may keep track of students’ fees and their payments, each user maintains separate files— and programs to manipulate these files This redundancy in defining and storing data results in wasted storage space and in redundant efforts to maintain common up-to-date data.

In the database approach, a single repository maintains data that is defined once and then accessed by various users.

The main characteristics of the database approach versus the file-processing approach are the following:

1. *Self-describing nature of a database system*
2. *Insulation between programs and data, and data abstraction*
3. *Support of multiple views of the data*
4. *Sharing of data and multiuser transaction processing*
5. **Self-Describing Nature of a Database System**

A fundamental characteristic of the database approach is that the database system contains not only the database itself but also a complete definition or description of the database structure and constraints. This definition is stored in the DBMS catalog, which contains information such as the structure of each file, the type and storage format of each data item, and various constraints on the data. The information stored in the catalog is called **meta-data**, and it describes the structure of the primary database (Figure 1.1). The catalog is used by the DBMS software and also by database users who need information about the database structure

In traditional file processing, data definition is typically part of the application programs themselves. Hence, these programs are constrained to work with only *one specific database,* whose structure is declared in the application programs. Forexample, an application program written in C++ may have struct or class declarations,file-processing software can access only specific databases,

DBMS softwarecan access diverse databases by extracting the database definitions from the catalogand using these definitions.

1. **Insulation between Programs and Data, and Data Abstraction**

In traditional file processing, the structure of data files is embedded in the application programs, so any changes to the structure of a file may require *changing all programs* that access that file. By contrast, DBMS access programs do not require such changes in most cases. The structure of data files is stored in the DBMS catalog separately from the access programs.We call this property **program-data independence**

1. **Support of Multiple Views of the Data**

A database typically has many users, each of whom may require a different perspective or **view** of the database. A view may be a subset of the database or it may contain **virtual data** that is derived from the database files but is not explicitly stored. Some users may not need to be aware of whether the data they refer to is stored or derived. A multiuser DBMS whose users have a variety of distinct applications must provide facilities for defining multiple views.

1. **Sharing of Data and Multiuser Transaction Processing**

A multiuser DBMS, as its name implies, must allow multiple users to access the database at the same time. The DBMS includes **concurrency** **control** software to ensure that several users trying to update the same data do so in a controlled manner so that the result of the updates is correct

**Q) ADVANTAGES OF USING DATABASE APPROACH**

1. **Controlling Redundancy**

In traditional software development utilizing file processing, every user group maintains its own files for handling its data-processing applications.

This **redundancy** in storing the same data multiple times leads to several problems.

* *Duplication of data*
* Wastage of storage space
* Inconsistency of data

Database approach helps overcome all the above problems by integrating data of different user groups at a single place

1. **Restricting Unauthorized Access**

When multiple users share a large database, most users will not be authorized to access all information in the database. For example, financial data is often considered confidential and only authorized persons are allowed to access such data. In addition, some users may only be permitted to retrieve data, whereas others are allowed to retrieve and update. Hence, the type of access operation— retrieval or update—must also be controlled. Typically, users or user groups are given account numbers protected by passwords, which they can use to gain access to the database. Thus DBMS provides **security and authorization subsystem**.

1. **Providing Persistent Storage for Program Objects**

Databases can be used to provide **persistent storage** for program objects and data structures. This is one of the main reasons for **object-oriented database systems**. Object-oriented database systems typically offer data structure **compatibility** with one or more object-oriented programming languages.

1. **Providing Storage Structures and Search Techniques for Efficient Query Processing**

Database is typically stored on disk, the DBMS providespecialized data structures and search techniques to speed up disk search for thedesired records. Auxiliary files called **indexes** are used for this purpose.

1. **Providing Backup and Recovery**

A DBMS must provide facilities for recovering from hardware or software failures. The **backup and recovery subsystem** of the DBMS is responsible for recovery. For example, if the computer system fails in the middle of a complex update transaction, the recovery subsystem is responsible for making sure that the database is restored to the state it was in before the transaction started executing. Disk backup is also necessary in case of a catastrophic disk failure.

1. **Providing Multiple User Interfaces**

Different types of users with varying levels of technical knowledge use a database, a DBMS provides a variety of user interfaces.

1. **Representing Complex Relationships among Data**

A database may include numerous varieties of data that are interrelated in many ways. A DBMS has the capability to represent a variety of complex relationships among the data and to define new relationships as they arise.

1. **Enforcing Integrity Constraints**

Most database applications have certain **integrity constraints** that must hold for the data. A DBMS provide capabilities for defining and enforcing

* **referential integrity** constraint
* **key** or **uniqueness** constraint etc...

1. **Permitting Inferencing and Actions Using Rules**

Database systems provide capabilities for defining *deduction rules* for *inferencing* new information from the stored database facts.

1. **Reduced Application Development Time**
2. **Flexibility**
3. **Availability of Up-to-Date Information**

**DISADVANTAGES OF DBMS**

1. **Complexity**
2. **Size**
3. **Performance**
4. **High impact of failure**
5. **Cost**
6. **Additional H/w cost**
7. **Cost of Conversion**

**Q) List someDatabase-System Applications**

1. ***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 and orders for items
* *Online retailers*: online order tracking, generation of recommendation lists, and maintenance of online product evaluations.

*2.* ***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.

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

4) ***Airlines***: For reservations and schedule information. Airlines were among the

first to use databases in a geographically distributed manner.

5)***Telecommunication*:** For keeping records of calls made, generating monthly

bills, maintaining balances on prepaid calling cards, and storing information

about the communication networks

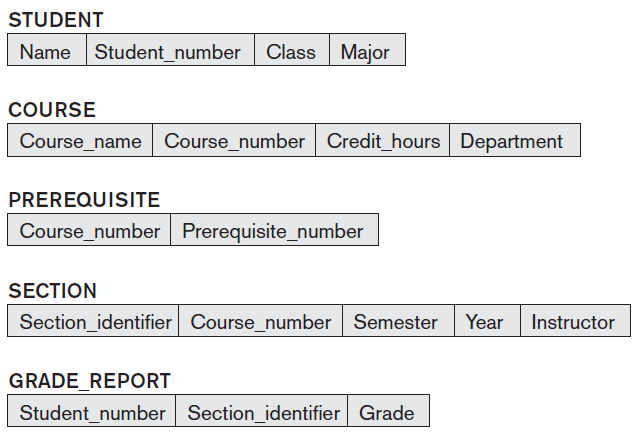
**Q) Define Database schema, Database instance, What is the need for DB schema, Example of schema**

***Q) Define* Instances and Database State**

**Q) Difference between Database schema and Database state**

**Database Schema**

The description of a database is called the **database schema**, which is specified during database design and is not expected to changefrequently. A displayed schema is called a **schema diagram**. Figure shown below shows a schema diagram, the diagram displays the structure of each record type but not the actual instances of records. We call each object in the schema—such as STUDENT or COURSE—a **schema construct**.



The actual data in a database may change quite frequently. For example, the database shown in Figure 1.2 changes every time we add a new student or enters 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** in the database. When we define a new database, we specify its database schema only to the DBMS. 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 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*.

***Q) Define* Meta-Data, Intension, Extension, Schema evolution.**

DBMS stores the descriptions of the schema constructs and constraints—also called the **meta-data**—in the **DBMS catalog** so that DBMS software can refer to the schema whenever it needs to.

The schema is sometimes called the **intension**, and a database state is called an **extension** of the schema.

The schema may need to be changed occasionally. For example, we may decide that another data item needs to be stored for each record in a file, such as adding the Date\_of\_birth to the STUDENT schema. This is known as **schema evolution**

**Q) Categorize the data models/ Explain Data Model and its significance.**

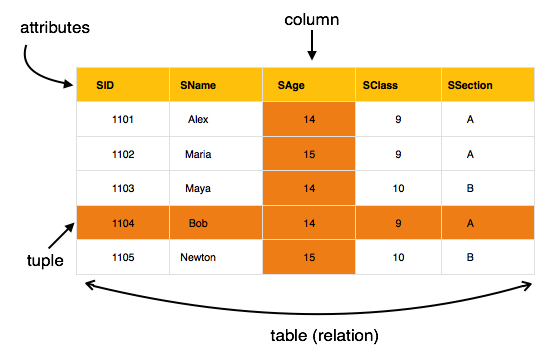
**DATA MODELS**

A **data model** is a collection of concepts that can be used to describe the structure of a database. It provides the necessary means to achieve data abstraction. **Data abstraction** generally refers to hiding of details of data organization and storage, and highlighting only the essential features for an improved understanding of data. One of the main characteristics of the database approach is to support data abstraction so that different users can perceive data at their preferred level of detail. Most data models also include a set of **basic operations** for specifying retrievals and updates on the database.

**Categories of DATA MODELS**

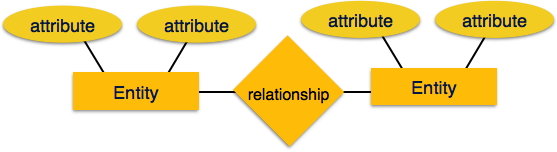
We can categorize Data Models according to the types of concepts they use to describe the database structure.

* **High-level** or **conceptual data models** provide concepts that are close to the way many users perceive data.
* **Low-level** or **physical data models** provide concepts that describe the details of how data is stored on the computer storage media, typically magnetic disks. Concepts provided by low-level data models are generally meant for computer specialists, not for end users.
* Between these two extremes is a class of **representational** (or **implementation**) **data models**, which provide concepts that may be easily understood by end users but that are not too far removed from the way data is organized in computer storage.
* **Relational Model** uses a collection of tables to represent both data and the relationships among data. Each table has multiple columns and each column has a unique name.



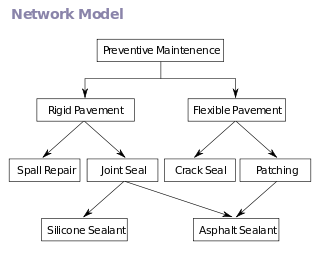
The main highlights of this model are −

* Data is stored in tables called **relations**.
* Relations can be normalized.
* In normalized relations, values saved are atomic values.
* Each row in a relation contains a unique value.
* Each column in a relation contains values from a same domain.
* **Entity-Relationship model-**  E-R model is based on the perception of real world that consists of a collection of basic objects called entities (objects in real world ) and relationship among these objects.

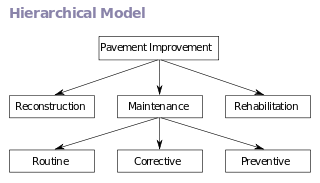


* **Object-Based Data Model** can be seen as extending the E-R model with notions of encapsulation, methods (functions) and object identity.

* **Network data model (**organizes data using two fundamental constructs, called records and sets) The **network model** is a [database model](https://en.wikipedia.org/wiki/Database_model) conceived as a flexible way of representing objects and their relationships. Its distinguishing feature is that the schema, viewed as a graph in which object types are nodes and relationship types are arcs, is not restricted to being a hierarchy or [lattice](https://en.wikipedia.org/wiki/Lattice_graph).



**Hierarchical data model:** A **hierarchical database model** is a [data model](https://en.wikipedia.org/wiki/Data_model) in which the data is organized into a [tree](https://en.wikipedia.org/wiki/Tree_data_structure)-like structure. The data is stored as **records** which are connected to one another through **links**. A record is a collection of fields, with each field containing only one value. The **entity type** of a record defines which fields the record contains.

[](https://en.wikipedia.org/wiki/File:Hierarchical_Model.svg)

Example of a hierarchical model

A record in the hierarchical database model corresponds to a row (or tuple) in the [relational database model](https://en.wikipedia.org/wiki/Relational_database_model) and an entity type corresponds to a table (or relation).

The hierarchical database model mandates that each child record has only one parent, whereas each parent record can have one or more child records. In order to retrieve data from a hierarchical database the whole tree needs to be traversed starting from the root node.

**Semistructured Data Model**. The semistructured data model permits the specification of data where individual data items of the same type may have different sets of attributes. This is in contrast to the above data models, where every data item of a particular type must have the same set of attributes. The **Extensible Markup Language (XML)** is widely used to represent semistructured data.

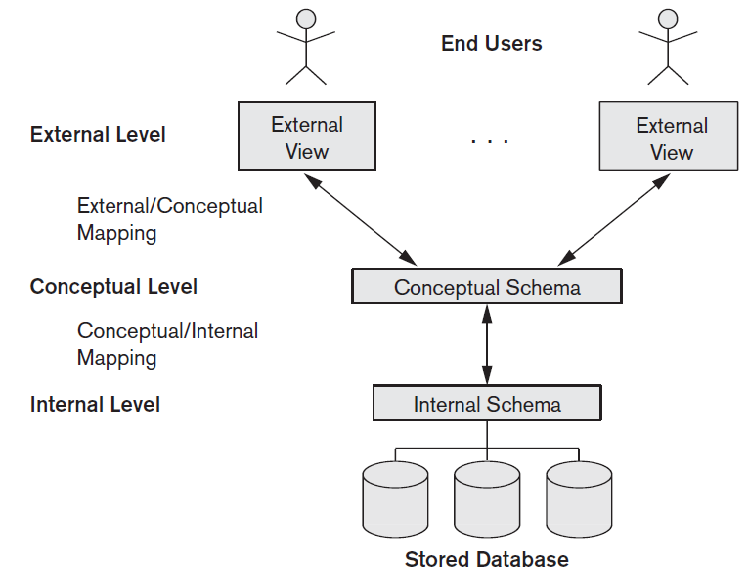
**Q) Illustrate the 3 schema architecture, explain role of each level and need for mapping.**

**Q) Explain how Database system provide data abstraction / explain diff levels of data abstraction (answer by explaining 3 schema architecture with each level and data independence)**

**THREE- SCHEMA ARCHITECTURE**

The goal of the three-schema architecture, illustrated in Figure 1.5 (next page), is to separate the user applications from the physical database. In this architecture, schemas can be defined at the following three levels (different levels of abstraction)

1. Internal Level/Physical Level
2. Conceptual level/Logical level
3. External level/ View level



**Three Schema Architecture**

1. The **internal level/physical level** has an **internal schema**, which describes the physical storage structure of the database, ie:- how the data are actually stored. The internal schema uses a physical data model and describes the complete details of data storage and access paths for the database
2. **Conceptual level/logical level** has a **conceptual schema** describes what data are stored in the database and what relationship exist among those data. It describes the structure of the whole database for a community of users. The conceptual 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 conceptual schema when a database system is implemented.
3. **External/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 level, each external schema is typically implemented using a representational data model. This system may provide many views for the same database.

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

**Q) Define Data Independence, explain two types of data independence.**

**Q) Differentiate between logical and physical data independence, which is harder to achieve**

**DATA INDEPENDENCE**

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** 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). After the conceptual schema undergoes a logical reorganization, application programs that reference the external schema constructs must work as before.
* **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 were reorganized—for example, by creating additional access paths—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. For example, providing an access path to improve retrieval speed should not require a query to be changed.

Data independence occurs because when the schema is changed at some level, the schema at the next higher level remains unchanged; only the *mapping* between the two levels is changed. Hence, application programs referring to the higher-level schema need not be changed.

Physical data independence exists in most databases and file environments where physical details such as the exact location of data on disk, and hardware details of storage are hidden from the user. Applications remain unaware of these details. On the other hand**, logical data independence is harder to achieve** because it allows structural and constraint changes without affecting application programs—a much stricter requirement

The three-schema architecture can make it easier to achieve true data independence, both physical and logical. However, the two levels of mappings create an overhead during compilation or execution of a query or program, leading to inefficiencies in the DBMS. Because of this, few DBMSs have implemented the full three schema architecture.

**Q) Categorize/Explain the various types of Database languages**

**DATABASE LANGUAGES**

**Database Languages**

A database system provides a **data-definition language** to specify the database schema and a **data-manipulation language** to express database queries and updates.

**Data-Manipulation Language**

A **data-manipulation language (DML)** is a language that enables users to access

or manipulate data as organized by the appropriate data model. The types of access are:

• Retrieval of information stored in the database

• Insertion of new information into the database

• Deletion of information from the database

• Modification of information stored in the database

**example of DML**

**select** *instructor*.*name*

**from** *instructor*

**where** *instructor*.*dept name* = ’Computer Science’

**There are basically two types of DML:**

• **Procedural DMLs** require a user to specify *what* data are needed and *how* to

get those data.

• **Declarative DMLs** (also referred to as **nonprocedural DMLs**) require a user to

specify *what* data are needed *without* specifying how to get those data.

**Data-Definition Language**

We specify a database schema by a set of definitions expressed by a special language called a **data-definition language** (**DDL**). The DDL is also used to specify

additional properties of the data. We specify the storage structure and access methods used by the database system by a set of statements in a special type of DDL called a **data storage and** **definition** language. These statements define the implementation details of the database schemas, which are usually hidden from the users. The data values stored in the database must satisfy certain **consistency constraints**. Access permission and authorisation can be set using DDL.

**example of DDL**

**create table** *department*

(*dept name* **char** (20),

*building* **char** (15),

*budget* **numeric** (12,2));

The output of the DDL is

placed in the **data dictionary**,which contains **metadata**—that is, data about data

SDL - Storage definition language is used to specify internal schema

VDL – View Definition language is used to specify user views and their mapping to conceptual schema.

**Q) Explain the various types of Database Interfaces**

**INTERFACES**

**Menu-Based Interfaces for Web Clients or Browsing.** These interfaces present

the user with lists of options (called **menus)** that lead the user through the formulation of a request

**Forms-Based Interfaces.** A forms-based interface displays a form to each user.

Users can fill out all of the **form** entries to insert new data, or they can fill out only

certain entries, in which case the DBMS will retrieve matching data for the remaining

entries

**Graphical User Interfaces.** A GUI typically displays a schema to the user in diagrammatic form. The user then can specify a query by manipulating the diagram. In many cases, GUIs utilize both menus and forms

**Natural Language Interfaces.** These interfaces accept requests written in English or some other language and attempt to *understand* them. The natural language

interface refers to the words in its schema, as well as to the set of standard words in

its dictionary, to interpret the request.

**Speech Input and Output.**

Applications with limited vocabularies such as inquiries for telephone directory,

flight arrival/departure, and credit card account information are allowing speech

for input and output to enable customers to access this information. The speech

input is detected using a library of predefined words and used to set up the parameters that are supplied to the queries. For output, a similar conversion from text or numbers into speech takes place.

**Interfaces for Parametric Users.** Parametric users, such as bank tellers, often

have a small set of operations that they must perform repeatedly.

**Interfaces for the DBA.** Most database systems contain privileged commands

that can be used only by the DBA staff. These include commands for creating

accounts, setting system parameters, granting account authorization, changing a

schema, and reorganizing the storage structures of a database

**Q) Categorize the people who handle/use Database - Database Users**

**Q) Explain the role/functions of Database administrator (DBA), Database designers**

**ACTORS OF SCENE** – People who handle / use Database

* people whose jobs involve the day-to-day use of a large database - we call them the *actors on the scene*

1. **Database Administrator**

Is a person who has central control of both the data and the programs that access those data.

**Functions of Database Administrator** (DBA)

1. **Schema Definition** – the DBA creates the original database schema by executing a set of data definition statements in DDL.
2. **Storage structure and access-method definition**
3. **Schema and physical organisation modification** - the DBA carries out changes to the schema and physical organisation to reflect the changing needs of the organisation or to alter the physical organisation to improve performance.
4. **Granting of authorization for data access** – DBA can regulate which parts of database various users can access.
5. **Routine Maintenance** – DBA maintenance activities include

* Periodically backing up the database
* Ensuring and upgrading disk space as required
* Ensuring performance is not degraded.

1. **Database Designers**

* **Database designers** are responsible for identifying the data to be stored in the database and for choosing appropriate structures to represent and store this data.
* It is the responsibility of database designers to communicate with all prospective database users in order to understand their requirements and to create a design that meets these requirements.

1. **Database Users**

**End users** are the people whose jobs require access to the database for querying, updating, and generating reports; the database primarily exists for their use. There are several categories of end users:

**Naive or parametric users** - 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

**Sophisticated end 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

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

**System Analysts and Application Programmers (Software Engineers)**

**System analysts** determine the requirements of end users, **Application programmers** implement these specifications as programs; then they test, debug, document, and maintain them.

**Workers Behind Scene**

* **DBMS system designers and implementers**
* **Tool developers**
* **Operators and maintenance personnel**

**Q) Explain/Categorize the various DBMS architectures**

**Q) Explain/ Diffrentiate two-tier and three tier architeture**

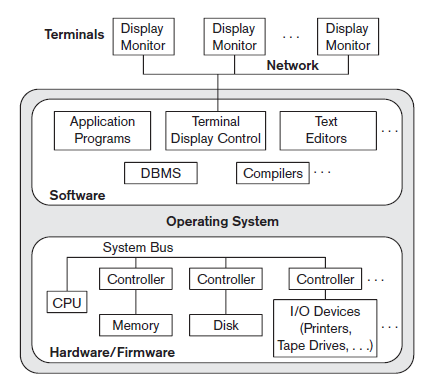
**Q) Explain the Client-server architecture for DBMS**

**DATABASE ARCHITECTURES**

The architecture of a database system is greatly influenced by the underlying computer system on which the database system runs. Database systems can be

1. centralized
2. client-server, where one server machine executes work on behalf of multiple client machines.
3. Database systems can also be designed to exploit parallel computer architectures.
4. Distributed databases span multiple geographically separated machines.
5. **Centralized DBMSs Architecture**

* 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.
* DBMS itself was **centralized** DBMS in which all the DBMS functionality, application program execution, and user interface processing were carried out on one machine



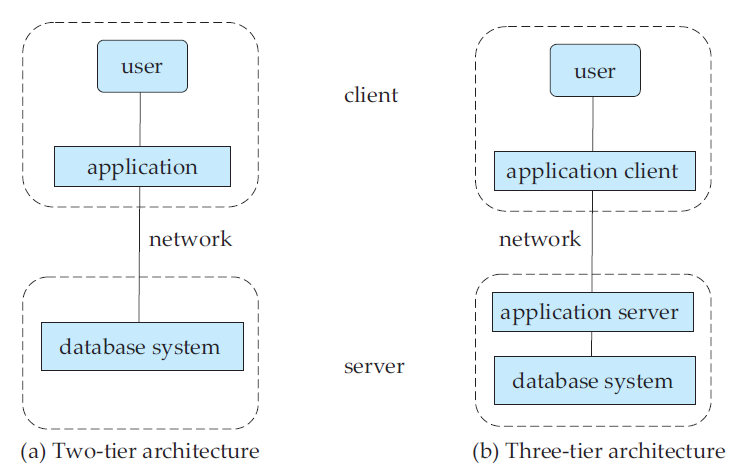
**Centralized DBMS Architecture**

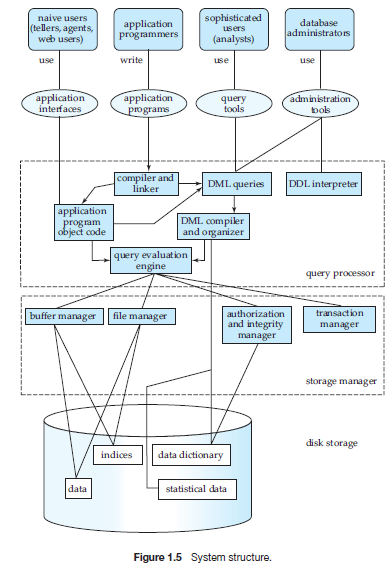
1. **Basic Client/Server Architectures**

* A **client** in this framework is typically a user machine that provides user interface capabilities and local processing. When a client requires access to additional functionality—such as database access—that does not exist at that machine, it connects to a server that provides the needed functionality.
* A **server** is a system containing both hardware and software that can provide services to the client machines, such as file access, printing, archiving, or database access.
* However, it is more common that client and server software usually run on separate machines
* Two main Client-Server Architecture

1. Two-Tier
2. Three-Tier

* In a **two-tier architecture**, the application resides at the client machine, where it invokes database system functionality at the server machine through query language statements. Application program interface standards like ODBC and JDBC are used for interaction between the client and the server.
* in a **three-tier architecture**, the client machine acts as merely a front end and does not contain any direct database calls. Instead, the client end communicates with an application server, usually through a forms interface. The application server in turn communicates with a database system to access data. The business logic of the application, which says what actions to carry out under what conditions, is embedded in the application server, instead of being distributed across multiple clients. Three-tier applications are more appropriate for large applications, and for applications that run on the WorldWideWeb.

****

****

**Q) Classifiy Database Management Systems**

1. The first is the **data model** on which the DBMS is based. The main data model used in many current commercial DBMSs is the **relational data model**. The **object data model** has been implemented in some commercial systems but has not had widespread use.Many legacy applications still run on database systems based on the **hierarchical** and **network data models.** Thus We can categorize DBMSs based on the data model: relational, object, object-relational, hierarchical,network, and other.
2. The second criterion used to classify DBMSs is the **number of users** supported by the system. **Single-user systems** support only one user at a time and are mostly used with PCs. **Multiuser systems**, which include the majority of DBMSs, support concurrent multiple users.
3. The third criterion is the **number of sites** over which the database is distributed. A DBMS is **centralized** if the data is stored at a single computer site. A centralized DBMS can support multiple users, but the DBMS and the database reside totally at a single computer site. A **distributed** DBMS (DDBMS) can have the actual database and DBMS software distributed over many sites, connected by a computer network. **Homogeneous** DDBMSs use the same DBMS software at all the sites, whereas **heterogeneous** DDBMSs can use different DBMS software at each site. It is also possible to develop **middleware software** to access several autonomous preexisting databases stored under heterogeneousDBMSs. This leads to a **federated** DBMS (or **multidatabase system**), in which the participating DBMSs are loosely coupled .
4. The fourth criterion is **cost**. It is difficult to propose a classification of DBMSs based on cost. Today we have **open source (free)** DBMS products like MySQL and PostgreSQL that are supported by third-party vendors with additional services. then there are **licensed** products
5. We can also classify a DBMS on the basis of the **types of access path** options for storing files
6. DBMS can be **general purpose** or **special purpose**.When performance is a primary consideration, a special-purpose DBMS can be designed and built for a specific application; such a system cannot be used for other applications without major changes. Example - airline reservations and telephone directory systems

DATA MODELING USING **ENTITY-RELATIONSHIP MODEL**

**USING HIGH-LEVEL CONCEPTUAL DATA MODEL FOR DATABASE DESIGN**

* **Main Phases of Database Design**

1. **Requirements collection and analysis**-During this step, the database designers interview prospective database users to understand and document their **data requirements**. The result of this step is a concisely written set of users’ requirements. These requirements should be specified in as detailed as possible.
2. In parallel with specifying the data requirements, it is useful to **specify the known functional requirements** of the application. These consist of the user defined **operations** (or **transactions**) that will be applied to the database, including both retrievals and updates.
3. Once the requirements have been collected and analyzed, the next step is to create a **conceptual schema** for the database, using a high-level conceptual data model. This step is called **conceptual design**. The conceptual schema is a concise description of the data requirements of the users and includes detailed descriptions of the entity types, relationships, and constraints; these are expressed using the concepts provided by the high-level data model (ex: Entity-Relationship Model (E-R model))

Q)  **Features/Advantages of such a High level/Conceptual design/E-R** **Model**

* These concepts/design do not include implementation details, they are usually easier to understand and can be used to communicate with nontechnical users/clients
* The high-level conceptual schema can also be used as a reference to ensure that all users’ data requirements are met and that the requirements do not conflict.
* This approach enables database designers to concentrate on specifying the properties of the data, without being concerned with storage and implementation details.
* During or after the conceptual schema design, the basic data model operations can be used to specify the high-level user queries and operations identified during functional analysis. This also serves to confirm that the conceptual schema meets all the identified functional requirements
* Modifications in the conceptual schema are possible as functional requirement changes are made.

1. The next step in database design is the actual implementation of the database, using a commercial DBMS. The conceptual schema is transformed from high level data model into implementation data model. This step is called **logic design or data model mapping** and its result is a database schema in the implementation data model.
2. The last step is the **physical design** phase, during which the internal storage structures, file organizations, indexes, access paths, and physical design parameters for the database files are specified. In parallel with these activities, application programs are designed and implemented

**AN EXAMPLE DATABASE APPLICATION illustrated using E-R Diagram**

We use sample database application, called COMPANY to illustrate the basic ER model concepts and their use in schema design.

The list of data requirements for the database are provided, we create its conceptual

schema step-by-step using the concepts of the ER model.

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 provide the following description of the *miniworld*

■ 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.

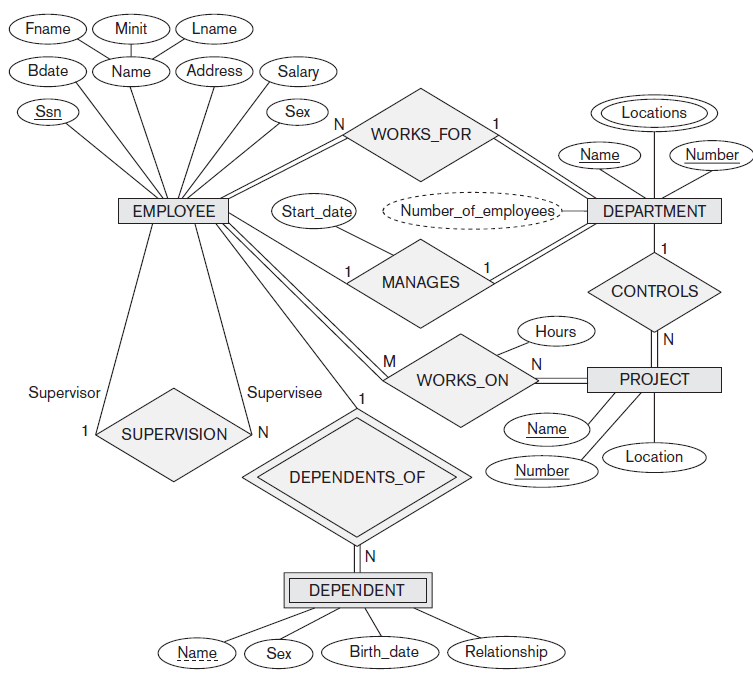
■ A department controls a number of projects, each of which has a unique name, a unique number, and a single location.

■ We store each employee’s name, Social Security number, address, salary, sex (gender), 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 current number of hours per week that an employee works on each project. We also keep track of the direct supervisor of each employee (who is another employee).

■ 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 (below) shows how the schema for this database application can be displayed by means of the graphical notation known as **ER diagrams**.

We can describe the step by- step process of deriving this schema from the stated requirements—and explain the ER diagrammatic notation—as we introduce the ER model concepts.



**Q) Explain E-R model with example (can summarize with example)**

**Q) Explain different types of attribute, what do you mean by NULL attribute when can an attribute value be NULL.**

**Q) Each type of attributes can be asked to be explained individually with example or Different types of Attributes with example or Comparison between any two attributes.**

**ENTITIES and ATTRIBUTES**

**Entity** - is a *thing* in the real world with an independent existence. An entity may be an object with a physical existence (for example, a particular person, car, house, or employee) or it may be an object with a conceptual existence (for instance, a company, a job, or a university course).

Each entity has **attributes**—the particular properties that describe it. For example, an EMPLOYEE entity may be described by the employee’s name, age, address, salary, and job. A particular entity will have a value for each of its attributes.

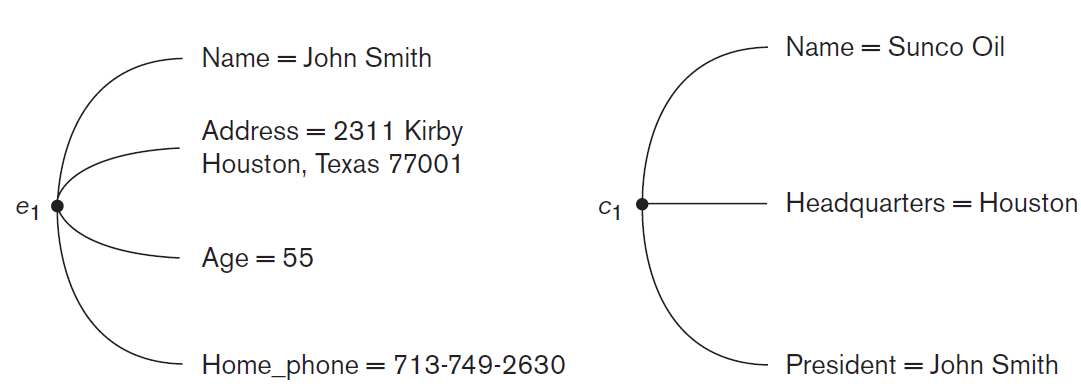


Figure1.7 (Two entities, EMPLOYEE e1, and COMPANY c1, and their attributes)

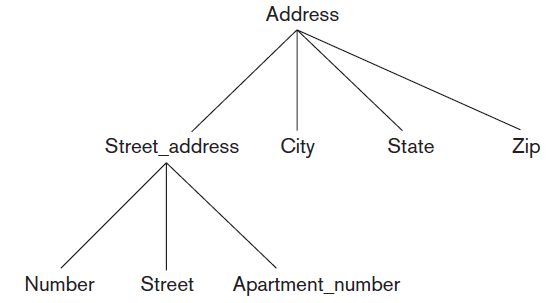
Types of attributes in the ER model:

1. *Simple* versus *composite*
2. *Singlevalued* versus *multivalued*
3. *Stored* versus *derived*.
4. *Complex*

1. **Composite versus Simple (Atomic) Attributes.**

**Composite attributes** can be divided into smaller subparts, which represent more basic attributes with independent meanings. For example, the Address attribute of the EMPLOYEE entity shown in Figure (below) can be subdivided into Street\_address, City, State, and Zip, with the values ‘2311 Kirby’, ‘Houston’, ‘Texas’, and ‘77001.’

Composite attributes can form a hierarchy; for example, Street\_address can be further subdivided into three simple component attributes: Number, Street, and Apartment\_number



Attributes that are not divisible are called **simple** or **atomic attributes**

The value of a composite attribute is the concatenation of the values of its component simple attributes.

Composite attributes are useful to model situations in which a user sometimes refers to the composite attribute as a unit but at other times refers specifically to its components. If the composite attribute is referenced only as a whole, there is no need to subdivide it into component attributes.

For example, if there is no need to refer to the individual components of an address (Zip Code, street, and so on), then the whole address can be designated as a simple attribute.

**2. Single-Valued versus Multivalued Attributes**

**Single-Valued Attributes** have a single value for a particular entity. For example, Age is a single-valued attribute of a person. Cars with one color have a **single value**

**Multivalued Attributes** can have a set of values for the same entity—for instance, Colors attribute for a car, two-tone cars have two color values or a College\_degrees attribute for a person.

One person may not have a college degree, another person may have one, and a third person may have two or more degrees; therefore, different people can have different *numbers* of *values* for the College\_degrees attribute. Such attributes are called **multivalued**.

A multivalued attribute may have lower and upper bounds to constrain the *number of values* allowed for each individual entity. For example, the Colors attribute of a car may be restricted to have between one and three values, if we assume that a car (single car) can have three colors at most.

**3. Stored versus Derived Attributes.**

In some cases, two (or more) attribute values are related—for example, the Age and Birth\_date attributes of a person. For a particular person entity, the value of Age can be determined from the current (today’s) date and the value of that person’s Birth\_date.

The Age attribute is hence called a **derived attribute** and is said to be **derivable from** the Birth\_date attribute, which is called a **stored attribute**.

Some attribute values can be derived from *related entities*; for example, an attribute Number\_of\_employees of a DEPARTMENT entity can be derived by counting the number of employees related to (working for) that department.

**NULL Values for attributes**

In some cases, a particular entity may not have an applicable value for an attribute. For such situations, a special value called NULL is created.

For example, the Apartment\_number attribute of an address applies only to addresses that are in apartment buildings and not to other types of residences, such as single-family homes.

Similarly, a College\_degrees attribute applies only to people with college degrees.

NULL can also be used if we do not know the value of an attribute for a particular entity—for example, if we do not know the home phone number of a person.

The meaning of the former type of NULL (ex Apartment no) is *not applicable*, whereas the meaning of the latter( ex home phone no) is *unknown*.

The *unknown* category ofNULL can be further classified into two cases.

1. The first case arises when it is knownthat the attribute value exists but is *missing*—for instance, ex: if the Height attribute of aperson is listed as NULL.
2. The second case arises when it is *not known* whether theattribute value exists—for example, if the Home\_phone attribute of a person is NULL.

**Complex Attributes.**

Composite and multivalued attributes can be nested arbitrarily. We can represent arbitrary nesting by grouping components of a **composite attribute between parentheses ( )** and separating the components with commas, and by displaying **multivalued attributes between braces { }**. Such attributes are called **complex attributes**.

For example, if a person can have more than one residence and each residence can have a single address and multiple phones, an attribute Address\_phone for a person can be specified as

{Address\_phone( {Phone(Area\_code, Phone\_number)},Address(Street\_address

(Number, Street, Apartment\_number), City, State, Zip) )}

Both Phone and Address are themselves composite attributes.

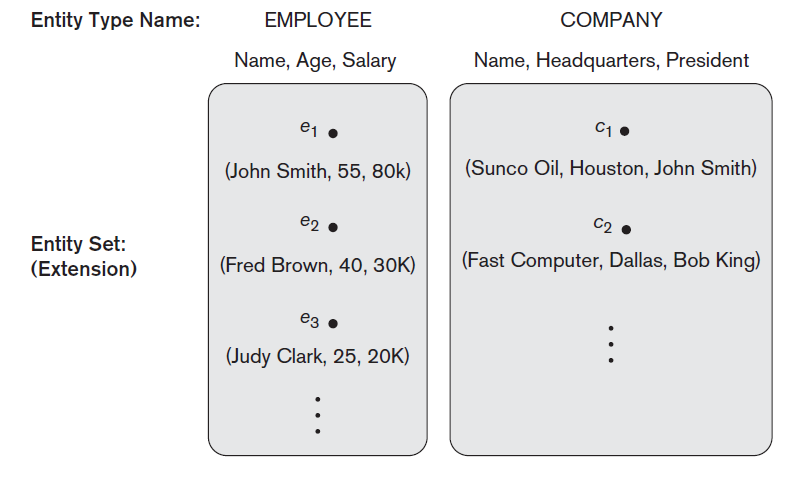
***Q) Define* Entity Types and Entity Sets.**

A database usually contains groups of entities that are similar. For example, a company employing hundreds of employees may want to store similar information concerning each of the employees.

These employee entities share the same attributes, but each entity has its *own value*(*s*) for each attribute. An **entity type** defines a *collection* (or *set*) of entities that have the same attributes. Each entity type in the database is described by its name and attributes.

The collection of all entities of a particular entity type in the database at any point in time is called an **entity set**; the entity set is usually referred to using the same name as the entity type.

Figure 1.9 (below)shows two entity types: EMPLOYEE and COMPANY, and a list of some of the attributes for each. A few individual entities of each type are also illustrated, along with the values of their attributes.

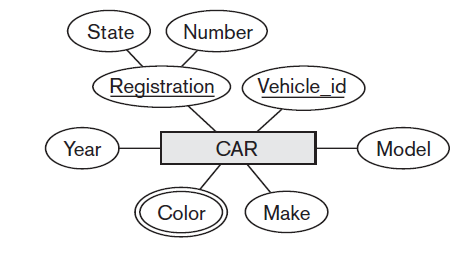


**Figure 1.9**

An **entity type is represented in ER diagram as a rectangular box** enclosing the entity type name. **Attribute names are enclosed in ovals** and are attached to their entity type by straight lines. Composite attributes are attached to their component attributes by straight lines. **Multivalued attributes are displayed in double ovals**.

(Also look figure Figure 1.6, example of COMPANY)

Figure 1.10 shows a CAR entity type in this notation.



**Figure 1.10**

An entity type describes the **schema** or **intension** for a *set of entities* that share the

same structure. The collection of entities of a particular entity type is grouped into

an entity set, which is also called the **extension** of the entity type.

**Q) Define Key Attribute, Composite Attribute of an Entity Type.**

An important constraint on the entities of an entity type is the **key** or **uniqueness constraint** on attributes. An entity type usually has one or more attributes whose values are distinct for each individual entity in the entity set. Such an attribute is called a **key attribute**, and its values can be used to identify each entity uniquely. Thus no two entities should have the same value for the key attribute at the same time. It is a constraint on *any entity set* of the entity type at any point in time

For example, the Name attribute is a key of the COMPANY entity type in Figure 1.9 because no two companies are allowed to have the same name.

For the PERSON entity type in Figure 1.6, a typical key attribute is Ssn (Social Security number).

In ER diagrammatic notation**, each key attribute has its name underlined** inside the oval

Sometimes several attributes together form a key, meaning that the *combination* of the attribute values must be distinct for each entity. If a set of attributes possesses this property, the proper way to represent this in the ER model that we describe here is to define a ***composite attribute***and designate it as a key attribute of the entity type

Some entity types have *more than one* key attribute. For example, each of the Vehicle\_id and Registration attributes of the entity type CAR (Figure 1.10) is a key in its own right. The Registration attribute is an example of a composite key formed from two simple component attributes, State and Number, neither of which is a key on its own.

An entity type may also have *no key*, in which case it is called a ***weak entity type***

**Q) Define Value Sets (Domains) of Attributes.**

Each simple attribute of an entity type is associated with a **value set** (or **domain** of values), which specifies the set of values that may be assigned to that attribute for each individual entity.

In Figure 1.9, if the range of ages allowed for employees is between 16 and 70, we can specify the value set of the Age attribute of EMPLOYEE to be the set of integer numbers between 16 and 70.

Similarly, we can specify the value set for the Name attribute to be the set of strings of alphabetic characters separated by blank characters, and so on.

Value sets are not displayed in ER diagrams, and are typically specified using the basic **data** **types** available in most programming languages, such as integer, string, Boolean, float, enumerated type and so on.

**Initial Conceptual Design of the COMPANY Database**

We can now define the entity types for the COMPANY database, based on the

Requirements, we can identify four entity types—one corresponding to each of the four items in the specification (look the specifications)

**1.** An entity type DEPARTMENT with attributes Name, Number, Locations,

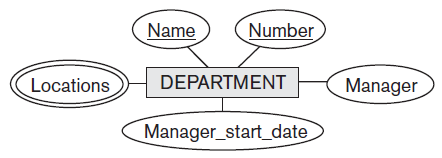
Manager, and Manager\_start\_date. Locations is the only multivalued attribute.

We can specify that both Name and Number are (separate) key attributes

because each was specified to be unique.

DEPARTMENT

Name, Number, {Locations}, Manager, ManagerStartDate

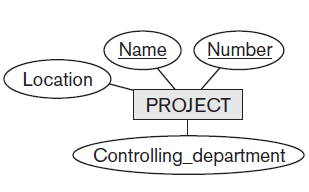


**2.** An entity type PROJECT with attributes Name, Number, Location, and

Controlling\_department. Both Name and Number are (separate) key attributes.

PROJECT

Name, Number, Location, ControllingDepartment



**3.** An entity type EMPLOYEE with attributes Name, Ssn, Sex, Address, Salary,

Birth\_date, Department, and Supervisor. Both Name and Address may be composite

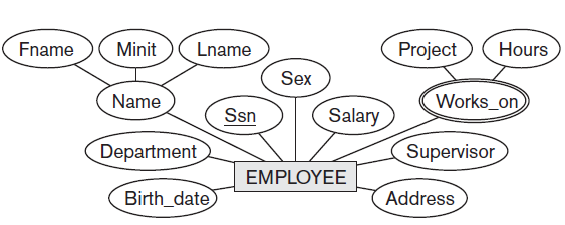
attributes; however, this was not specified in the requirements. We

must go back to the users to see if any of them will refer to the individual

components of Name—First\_name, Middle\_initial, Last\_name—or of Address.

EMPLOYEE

Name(FName, Minit, LName), Ssn, Sex, Addres, Salary, BirthDate, Department, Supervisor, {WorksOn(Project, Hours)}



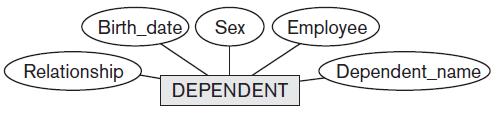
We represent the fact that an employee can work on several projects, and the number of hours per week an employee works on each project as a multivalued composite attribute of EMPLOYEE, called Works\_on with the simple components (Project, Hours).

**4.** An entity type DEPENDENT with attributes Employee, Dependent\_name, Sex,

Birth\_date, and Relationship (to the employee).

DEPENDENT

Employee, DependentName, Sex, BirthDate, Relationship



**Q) Define Relationships between Entities**

Whenever an attribute of one entity type refers to another entity type, some relationship exists.

For example, the attribute Manager of DEPARTMENT refers to an employee who manages the department; the attribute Controlling\_department of PROJECT refers to the department that controls the project; the attribute Supervisor of EMPLOYEE refers to another employee (the one who supervises this employee); the attribute Department of EMPLOYEE refers to the department for which the employee works; and so on.

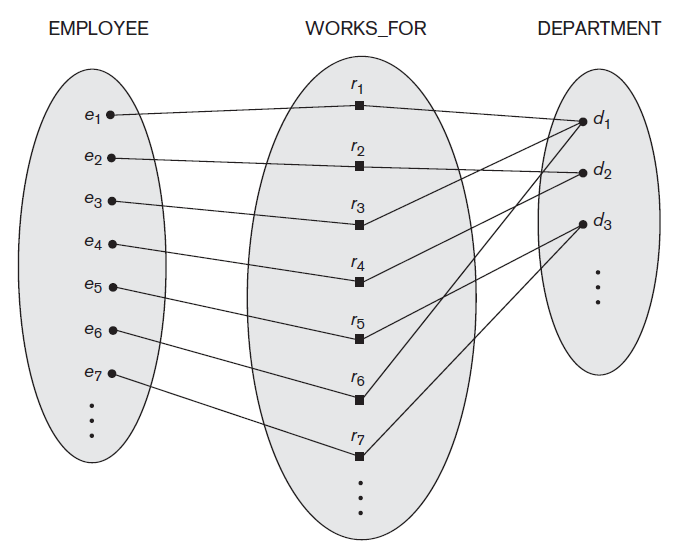
In the ER model, these references should not be represented as attributes but as **relationships**

In the initial design of entity types, relationships are typically captured in the form of attributes. As the design is refined, these attributes get converted into relationships between entity types.

**Q) Define/Differentiate Relationship type, sets and instances with example.**

A **relationship type** *R* among *n* entity types *E*1, *E*2, ..., *En* defines a set of associations—or a **relationship set**—among entities from these entity types. Mathematically, the relationship set *R* is a set of **relationship instances** *ri,* where each *ri* associates *n* individual entities (*e*1, *e*2, ..., *en*), and each entity *ej* in *ri* is a member of entity set *Ej*, 1 <= *j* <= *n*. each of the individual entities *e*1, *e*2, ..., *en* is said to **participate** in the relationship instance *ri* = (*e*1, *e*2, ..., *en*).

Consider a relationship type WORKS\_FOR between the two entity types EMPLOYEE and DEPARTMENT, which associates each employee with the department for which the employee works in the corresponding entity set. Each relationship instance in the relationship set WORKS\_FOR associates one EMPLOYEE entity and one DEPARTMENT entity.



*, Some instances in the WORKS\_FOR relationship set, which represents a relationship type WORKS\_FOR between EMPLOYEE and DEPARTMENT.*

Figure 1.11 illustrates this example, where each relationship instance *ri* is shown connected to the EMPLOYEE and DEPARTMENT entities that participate in *ri*. In the miniworld represented by Figure 1.11, employees *e*1, *e*3, and *e*6 work for department *d*1; employees *e*2 and *e*4 work for department *d*2; and employees *e*5 and *e*7 work for department *d*3.

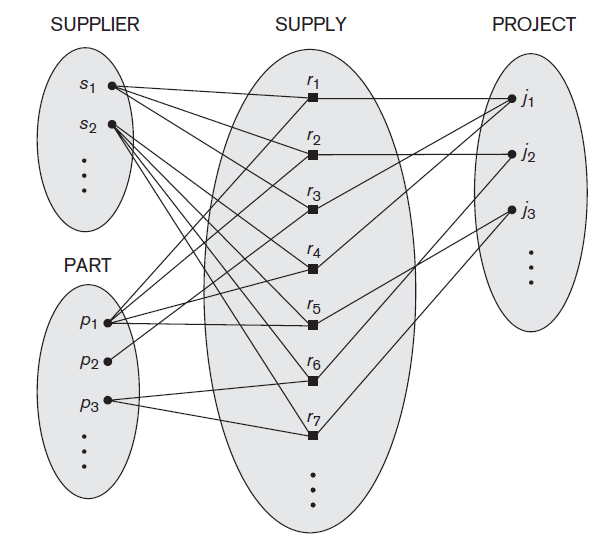
In ER diagrams**, relationship types are displayed as diamond-shaped boxes**, which are connected by straight lines to the rectangular boxes representing the participating entity types. The relationship name is displayed in the diamond-shaped box

**Q) Define Relationship Degree(Binary and Ternary relationships)**

The **degree** of a relationship type is the number of participating entity types.

Hence, the WORKS\_FOR (above) relationship is of degree two. A relationship type of degree two is called **binary**, and one of degree three is called **ternary**.

An example of a ternary relationship is SUPPLY, shown in Figure below where each relationship instance *ri* associates three entities—a supplier *s*, a part *p*, and a project *j*—whenever *s* supplies part *p* to project *j*. Relationships can generally be of any degree



**Ternary relationship**

**Define Role Names and Recursive Relationships.**

Each entity type that participates in a relationship type plays a particular role in the relationship. The **role name** signifies the role that a participating entity from the entity type plays in each relationship instance, and helps to explain what the relationship means. For example, in the WORKS\_FOR relationship type, EMPLOYEE plays the role of *employee* or *worker* and DEPARTMENT plays the role of *department* or *employer*.

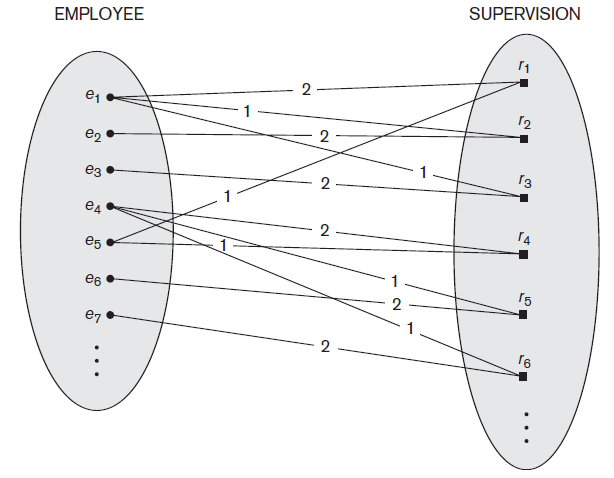
Role names are not technically necessary in relationship types where all the participating entity types are distinct, since each participating entity type name can be used as the role name.

However, in some cases the *same* entity type participates more than once in a relationship type in *different roles*. In such cases the role name becomes essential for distinguishing the meaning of the role that each participating entity plays. Such relationship types are called **recursive relationships**.

The SUPERVISION relationship type relates an employee to a supervisor, where both employee and supervisor entities are members of the same EMPLOYEE entity set. Hence, the EMPLOYEE entity type *participates twice* in SUPERVISION: once in the role of *supervisor* (or *boss*), and once in the role of *supervisee* (or *subordinate*).

Each relationship instance *ri* in SUPERVISION associates two employee entities *ej* and *ek*, one of which plays the role of supervisor and the other the role of supervisee. The lines marked ‘1’ represent the supervisor role, and those marked ‘2’ represent the supervisee role;

hence, *e*1 supervises *e*2 and *e*3, *e*4 supervises *e*6 and *e*7 (e4 plays the role of supervisor in relationship r5 and supervises e6(e4 to r5 ‘1’ and ‘2’ from r5 to e6) , and *e*5 supervises *e*1 and *e*4. In this example, each relationship instance must be connected with two lines, one marked with ‘1’ (supervisor) and the other with ‘2’ (supervisee).



**Figure 1.13**

**Q) Diff types of CONSTRAINTS ON RELATIONSHIP TYPES in E-R Model**

**Q) Explain Structural Constraint**

**Q) Expalin Cardinality ratio (diff types – 1:1, 1:N, M:N), Participation Constraint( diff types – total & partial) and Existence Dependencies**

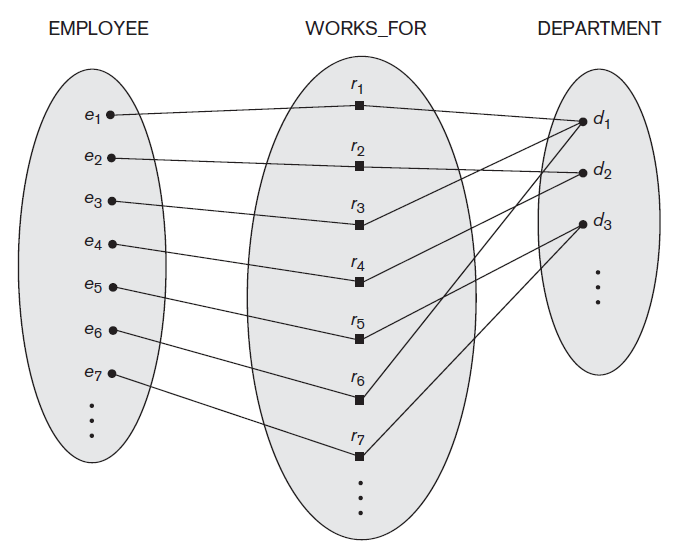
Relationship types usually have certain constraints that limit the possible combinations of entities that may participate in the corresponding relationship set ex: if the company has a rule that each employee must work for exactly one department

**Two main types of binary relationship constraints:**

* ***cardinality ratio* and**
* ***participation*.**

**Cardinality Ratios for Binary Relationships.**

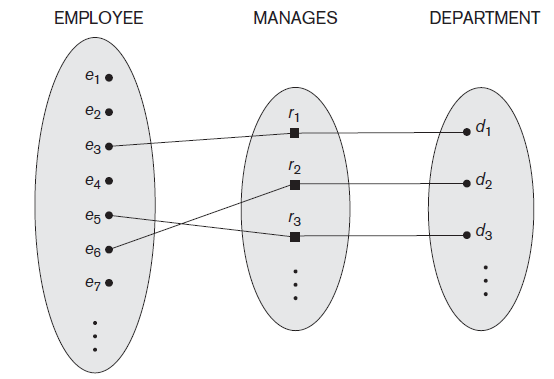
The **cardinality ratio** for a binary relationship specifies the *maximum* number of relationship instances that an entity can participate in.



**Figure 1.14**

Ex: in the WORKS\_FOR binary relationship type, DEPARTMENT:EMPLOYEE is of cardinality ratio 1:N, meaning that each department can be related to (that is, employs) any number of employees, but an employee can be related to (work for) only one department.

**The possible cardinality ratios for binary relationship types are 1:1, 1:N, N:1, and M:N.**

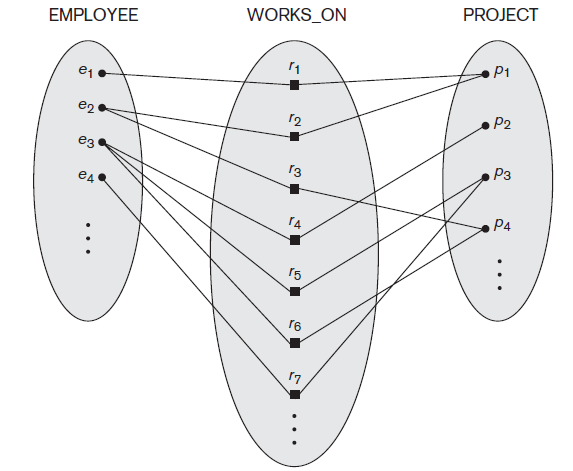


**Figure 1.15**

**An example of a 1:1 binary relationship is MANAGES** (Figure above 1.15),

which relates a department entity to the employee who manages that department. This represents the miniworld constraints that—at any point in time—an employee can manage one department only and a department can have one manager only.

,



**Figure 1.16**

The relationship type WORKS\_ON (Figure above 1.16) **is of cardinality ratio M:N** because the mini-world rule is that an employee can work on several projects and a project can have several employees.

**Cardinality ratios for binary relationships are represented on ER diagrams by displaying 1, M, and N on the diamonds** as shown in Figure 1.6.

In this notation, we can either specify no maximum (N) or a maximum of one (1) on participation. An alternative notation allows the designer to specify a specific *maximum number* on participation, such as 4 or 5.

**Participation Constraints and Existence Dependencies.**

The **participation constraint** specifies whether the existence of an entity depends on its being relatedto another entity via the relationship type.

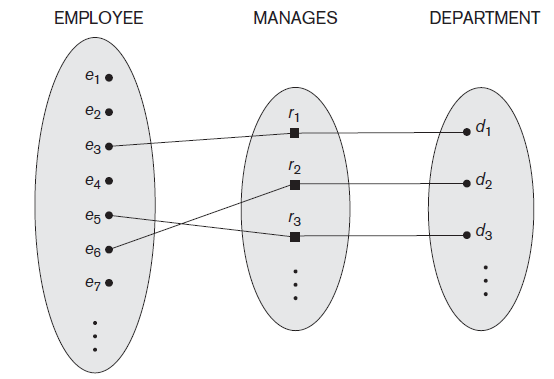
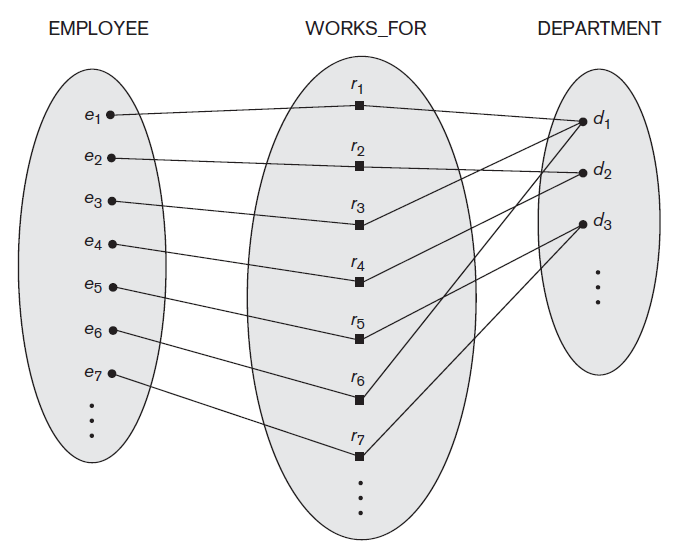
Ex: If a companypolicy states that *every* employee must work for a department, then an employeeentity can exist only if it participates in at least one WORKS\_FOR relationship instance (Figure below).

This constraint specifies the *minimum*number of relationship instances that each entity can participate in, and is sometimescalled the **minimum cardinality constraint**.

There are two types of participationconstraints—

1. total
2. partial

Thus, the participation of EMPLOYEE in WORKS\_FOR is called **total participation**, meaning that every entity in *the total set* of employee entities must be related to a department entity via WORKS\_FOR. ***Total participation is also called*** **existence dependency**.

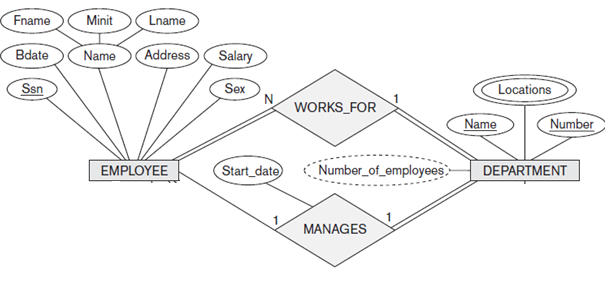


In Figure above we do not expect every employee to manage a department, so the participation of EMPLOYEE in the MANAGES relationship type is **partial**, meaning that *some* or *part of the set of* employee entities are related to some department entity via MANAGES, but not necessarily all.

**The cardinality ratio and participation constraints, taken together, as *the structural constraints* of a relationship type.**

In ER diagrams, **total participation (or existence dependency) is displayed as a**

***double line* connecting the participating entity type to the relationship, whereas partial participation is represented by a *single line***(see Figure below).



Notice that in this notation, we can either specify no minimum (partial participation) or a minimum of one (total participation). There is provision for designer to specify a specific *minimum number* on participation in the relationship, such as 4 or 5.

**Attributes of Relationship Types**

Relationship types can also have attributes, similar to those of entity types.

For example, to record the number of hours per week that an employee works on a particular project, we can include an attribute Hours for the WORKS\_ON relationship type look Figure 1.6. Another example is to include the date on which a manager started managing a department via an attribute Start\_date for the MANAGES relationship type look Figure 1.6.

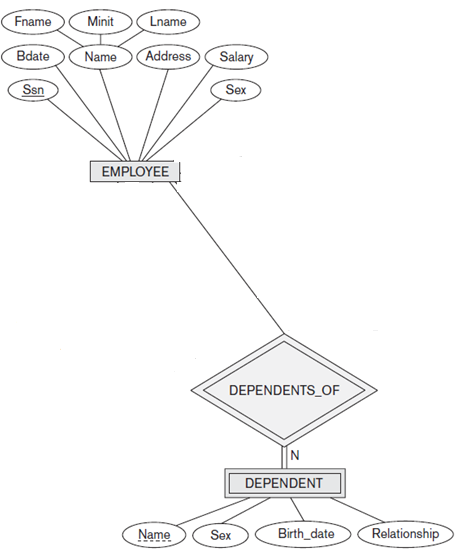
**Q) Define/Differentiate Weak Entity, Strong entity, Partial key, Identifying or owner entity**

Entity types that do not have key attributes 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.

Consider  **Figure below** the entity type **DEPENDENT (weak entity type),** related to **EMPLOYEE (identifying entity**), which is used to keep track of the dependents of each employee via a 1:N relationship .



In the above fig, the attributes of **DEPENDENT\_OF (identifying relation)** are Name (the first name of the dependent), Birth\_date, Sex, and Relationship (to the employee). Two dependents of *two distinct* *employees* may, by chance, have the same values for Name, Birth\_date, 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 attribute 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** (see Figure above). The **partial key attribute is underlined with a dashed or dotted line.**

In general, any number of levels of weak entity types can be defined; an owner entity type may itself be a weak entity type. In addition, a weak entity type may have more than one identifying entity type and an identifying relationship type of degree higher than two

**ADDING RELATIONSHIPS, CARDINALITY RATIO TO E-R DIAGRAMS**

We illustrated all requirements by users as attributes; we refine the database design in Intial Conceptual design (page 18) changing the attributes that represent relationships into relationship types. The cardinality ratio and participation constraint of each relationship type are determined from the requirements listed (page ). If some cardinality ratio or dependency cannot be determined from the requirements, the users must be questioned further to determine these structural constraints

We specify the following relationship types:

■ MANAGES, a 1:1 relationship type between EMPLOYEE and DEPARTMENT. EMPLOYEE participation is partial. DEPARTMENT participation is not clear from the requirements. We question the users, who say that a department must have a manager at all times, which implies total participation. The attribute Start\_date is assigned to this relationship type.

■ WORKS\_FOR, a 1:N relationship type between DEPARTMENT and EMPLOYEE. Both participations are total.

■ CONTROLS, a 1:N relationship type between DEPARTMENT and PROJECT. The participation of PROJECT is total, whereas that of DEPARTMENT is determined to be partial, after consultation with the users indicates that some departments may control no projects.

■ SUPERVISION, a 1:N relationship type between EMPLOYEE (in the supervisor role) and EMPLOYEE (in the supervisee role). Both participations are determined to be partial, after the users indicate that not every employee is a supervisor and not every employee has a supervisor.

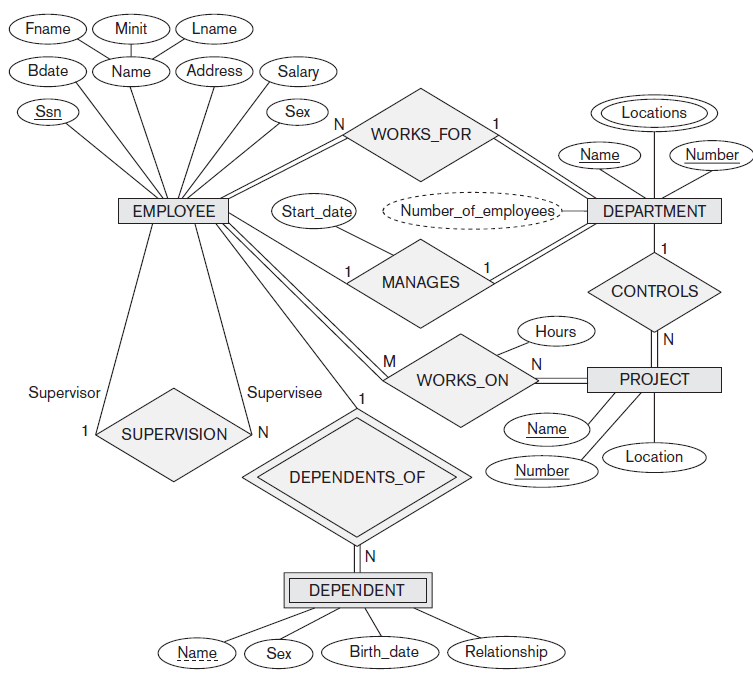
■ WORKS\_ON, determined to be an M:N relationship type with attribute Hours, after the users indicate that a project can have several employees working on it. Both participations are determined to be total.

■ DEPENDENTS\_OF, a 1:N relationship type between EMPLOYEE and DEPENDENT, which is also the identifying relationship for the weak entitytype DEPENDENT. The participation of EMPLOYEE is partial, whereas that of DEPENDENT is total.

After specifying the above six relationship types, we remove from the entity types in Intial Conceptual design shown in page , all attributes that have been refined into relationships. These include Manager and Manager\_start\_date from DEPARTMENT; Controlling\_department from PROJECT; Department, Supervisor, and Works\_on from EMPLOYEE; and Employee from DEPENDENT. It is important to have the least possible redundancy when we design the conceptual schema of a database

**Q) Explain various degrees of relationship types**

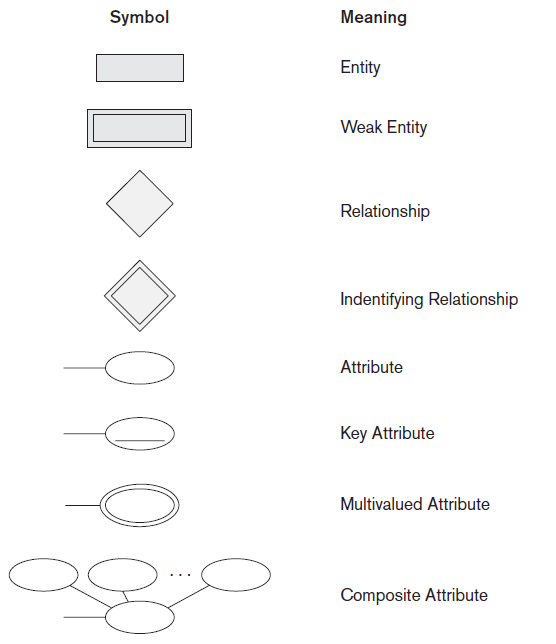
Degrees of relationships are 1:1 , 1:N and M:N (explain using example figures 1.14, 1.15 and 1.16 )

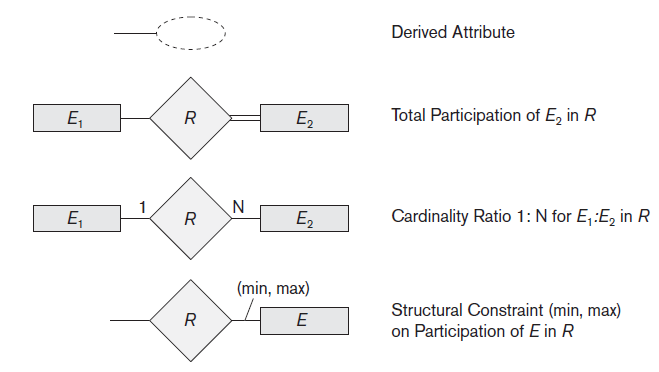


COMPANY **ER database schema** as an **ER diagram**

**(Complete Explanation)**

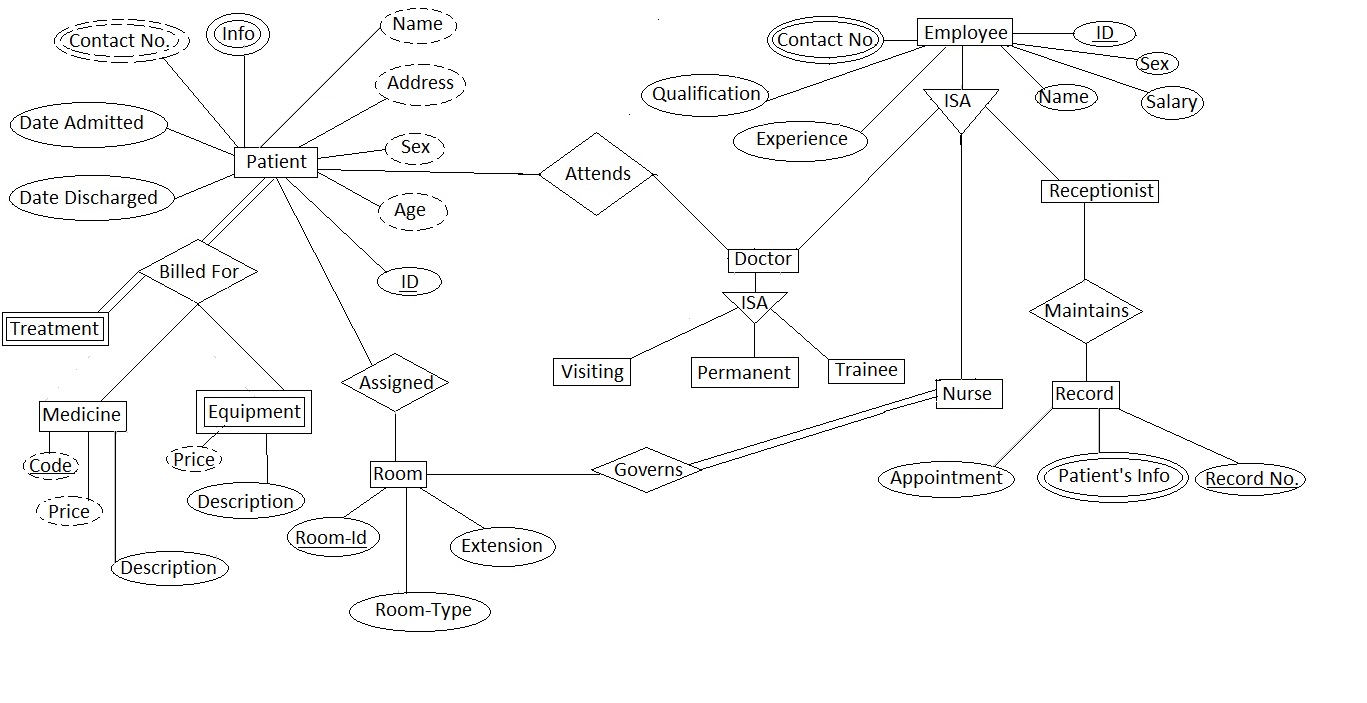
* Entity types such as EMPLOYEE, DEPARTMENT, and PROJECT are shown in rectangular boxes.
* Relationship types such as WORKS\_FOR, MANAGES, CONTROLS, and WORKS\_ON are shown in diamond-shaped boxes attached to the participating entity types with straight lines.
* Attributes are shown in ovals, and each attribute is attached by a straight line to its entity type or relationship type.
* Component attributes of a composite attribute are attached to the oval representing the composite attribute, as illustrated by the Name attribute of EMPLOYEE.
* Multivalued attributes are shown in double ovals, as illustrated by the Locations attribute of DEPARTMENT.
* Key attributes have their names underlined. Derived attributes are shown in dotted ovals, as illustrated by the Number\_of\_employees attribute of DEPARTMENT.
* Weak entity types are distinguished by being placed in double rectangles and by having their identifying relationship placed in double diamonds, as illustrated by the DEPENDENT entity type and the DEPENDENTS\_OF identifying relationship type.
* The partial key of the weak entity type is underlined with a dotted line.
* The cardinality ratio of each *binary* relationship type is specified by attaching a 1, M, or N on each participating edge. The cardinality ratio of DEPARTMENT:EMPLOYEE in MANAGES is 1:1, whereas it is 1:N for DEPARTMENT: EMPLOYEE in WORKS\_FOR, and M:N for WORKS\_ON.
* The participation constraint is specified by a single line for partial participation and by double lines for total participation (existence dependency).
* We show the role names for the SUPERVISION relationship type because the same EMPLOYEE entity type plays two distinct roles in that relationship. Notice that the cardinality ratio is 1:N from supervisor to supervisee because each employee in the role of supervisee has at most one direct supervisor, whereas an employee in the role of supervisor can supervise zero or more employees.



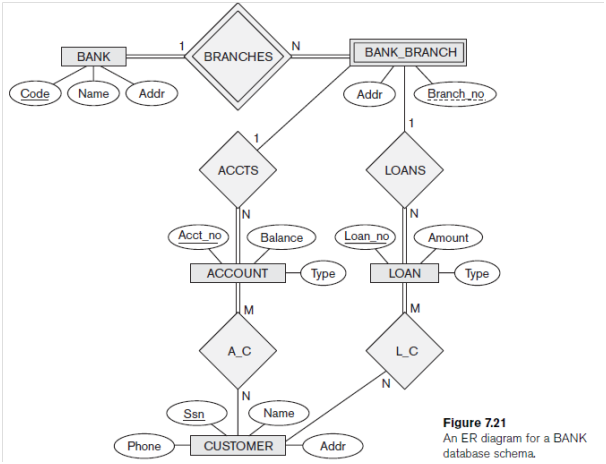


**MODULE 1 – Sample E-R’s**

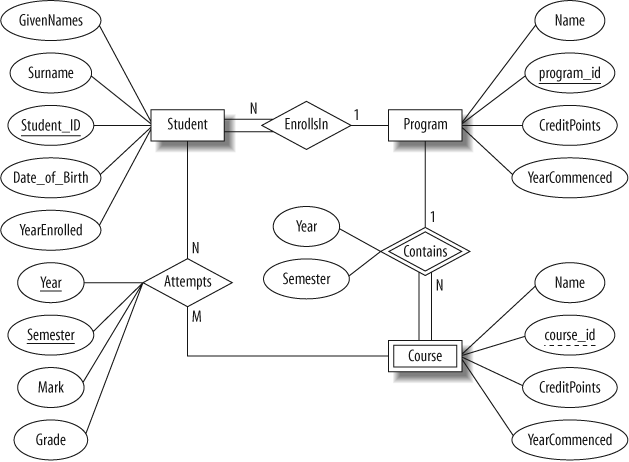
**Sample E-R diagram for Hospital (Participation and Cardinality not shown)**



**Sample E-R diagram for Bank showing loan and account details of a customer**



**Sample E-R showing a student joining and attempting for a course in a University/College**



**Sample Hospital-patient-Pharmacy E-R Diagram (dark line-total participation)**

