

# **ADVANCED DATABASE SYATEMS (22MCA102)**

**BY**

**Dr.Anantha Murthy  
Associate Professor  
Dept. of MCA  
NMAM.I.T, Nitte**



# Course Objectives

- **Understand the importance of DBMS and have thorough understanding of terminologies used.**
- **Implement concepts of relational model using SQL.**
- **Use the features of PL/SQL to write procedure programs.**
- **Design the database and to use different levels of Normalization.**
- **Understand the working of NoSQL, MongoDB and its features.**



# Books

## Text Books

- \* Elmasri and Navathe: Fundamentals of Database Systems, Seventh Edition 2016.
- \* Ivan Bayross : Commercial Application Development using Oracle Developer 2000.
- \* Seema acharya, Subhashini Chellappan, “Big Data Analytics”, 1<sup>st</sup> Edition, Wiley, 2015
- \* Raghu Ramakrishnan and Johannes Gehrke : Database Management Systems, Sixth Edition, Mcgraw-Hill.
- \* Joel Murach: Murach's MySQL, 3rd Edition 2019

## Reference Books :

- \* Silberschatz, Korth and Sudarshan: Database Systems Concepts, Sixth Edition, McGraw-Hill.
- \* Alexis Leon, Mathews Leon: Database Management Systems, Vikas Publishing House
- \* Connolly: Database Systems: A practical approach to design implementation and management, Third edition, Person Education.

# Unit 1

## Introduction to Database And Entity Relationship Model



# What is data, database, DBMS?


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

**Database:** A database is a collection of related data

**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 software package/ system to facilitate the creation and maintenance of a computerized database.


**Database System:** The DBMS software together with the data (database) itself. Sometimes, the applications are also included.





# What is data, database, DBMS? (Continued)

## Implicit properties of a Database:

- A database represents some aspect of the real world- miniworld or the universe of discourse (UoD). Changes to the miniworld are reflected in the database.
  - A database is a logically coherent collection of data with some inherent meaning. A random assortment of data cannot correctly be referred to as a database.
  - A database is designed, built, and populated with data for a specific purpose. It has an intended group of users and some preconceived applications in which these users are interested.
  - A database can be of any size and complexity.
  - A database may be generated and maintained manually, or it may be computerized.
- 




## What is data, database, DBMS? (Continued)

**A database management system (DBMS)** is a computerized system that enables users to create and maintain a database.

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

A typical large database may have a life cycle of many years, so the DBMS must be able to maintain the database system by allowing the system to evolve as requirements change over time.






## What is data, database, DBMS? (Continued)

**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.

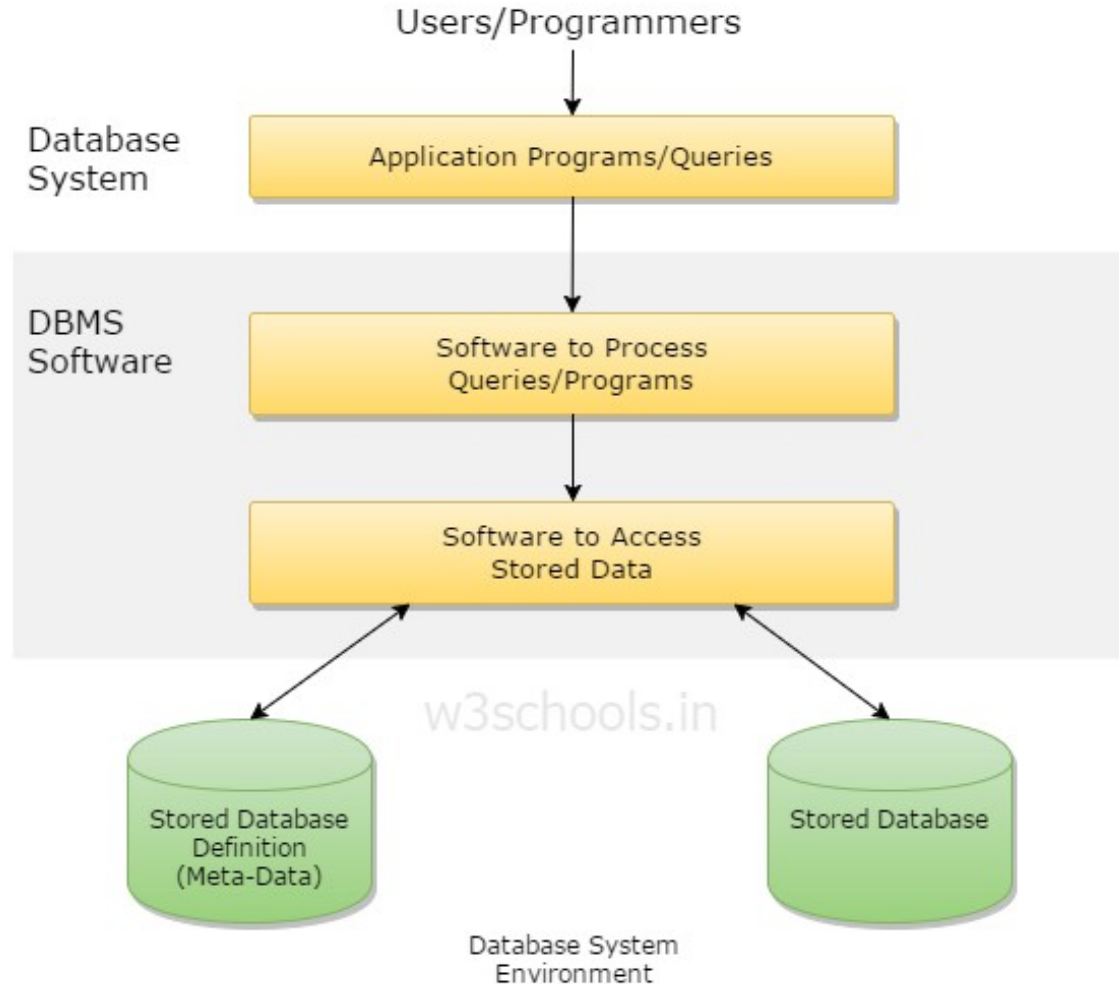
**Sharing** a database allows multiple users and programs to access the database simultaneously.





# A simplified Database System Environment


A database environment is a collective system of components that comprise and regulates the group of data, management, and use of data, which consist of software, hardware, people, techniques of handling database, and the data also.





# Characteristics of the Database Approach

## Self-Describing Nature of a Database System

- Database system contains complete definition or description of the database structure and constraints - stored in the *DBMS catalog*
  - Contains information such as the structure of each file, the type and storage format of each data item, and various constraints on the data.
  - Information stored in the catalog is called *metadata* - describes the structure of the primary database.
  - The catalog is used by the DBMS software and by database users who need information about the database structure.
  - *In traditional file processing, data definition is typically part of the application programs themselves*
- 

# Characteristics of the Database Approach (Continued)

## 1. Self-Describing Nature of a Database System

- Database system contains complete definition or description of the database structure and constraints - stored in the *DBMS catalog*
- Contains information such as the structure of each file, the type and storage format of each data item, and various constraints on the data.
- Information stored in the catalog is called *metadata* - describes the structure of the primary database.
- The catalog is used by the DBMS software and by database users who need information about the database structure.
- *In traditional file processing, data definition is typically part of the application programs themselves*

## *DBMS catalog & Metadata*

### RELATIONS

Relation_name	No_of_columns
STUDENT	4
COURSE	4
SECTION	5
GRADE_REPORT	3
PREREQUISITE	2

### COLUMNS

Column_name	Data_type	Belongs_to_relation
Name	Character (30)	STUDENT
Student_number	Character (4)	STUDENT
Class	Integer (1)	STUDENT
Major	Major_type	STUDENT
Course_name	Character (10)	COURSE
Course_number	XXXXNNNN	COURSE
....	....	....
....	....	....
....	....	....
Prerequisite_number	XXXXNNNN	PREREQUISITE

# Characteristics of the Database Approach (Continued)

## 2. 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 : **Program-data independence**

**Example:** Adding new column to a relation



# Characteristics of the Database Approach (Continued)

## 2. Insulation between Programs and Data, and Data Abstraction

In OO,OR systems, users can define *operations* on data as part of the database definitions.

User application programs can operate on the data by invoking these operations through their names and arguments, regardless of how the operations are implemented: **Program-operation independence**

The characteristic that allows these two properties: **Data abstraction**





# Characteristics of the Database Approach (Continued)

## 3. Support of Multiple Views of the Data


- A database typically has many types of 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.
- A multiuser DBMS whose users have a variety of distinct applications must provide facilities for defining multiple views.





## Characteristics of the Database Approach (Continued)

### 4. Sharing of Data and Multiuser Transaction Processing

- A multiuser DBMS must allow multiple users to access the database at the same time.
  - This is essential if data for multiple applications is to be integrated and maintained in a single database.
  - DBMS must include **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.
  - **Transaction** is an **executing program** or **process** that includes one or more database accesses.
- 



# Traditional File system Vs DBMS

Basis	File System	DBMS
1. Structure	File system is a software that manages and organizes the files in a storage medium within a computer.	DBMS is a software for managing the database.
2. Data Redundancy	Redundant data can be present in a file system.	In DBMS there is no redundant data.
3.Backup and Recovery	It doesn't provide backup and recovery of data if it is lost.	It provides backup and recovery of data even if it is lost.

## Traditional File system Vs DBMS (Continued)

Basis	File System	DBMS
4. Query processing	There is no efficient query processing in file system.	Efficient query processing is there in DBMS.
5. Consistency	There is less data consistency in file system.	There is more data consistency because of the process of normalization.
6. Complexity	It is less complex as compared to DBMS.	It has more complexity in handling as compared to file system.

## Traditional File system Vs DBMS (Continued)

Basis	File System	DBMS
7.Security Constraints	File systems provide less security in comparison to DBMS.	DBMS has more security mechanisms as compared to file system.
8.Cost	It is less expensive than DBMS.	It has a comparatively higher cost than a file system.
9. Data Independence	There is no data independence.	In DBMS data independence exists.



# Advantages of Using the Database Approach


## 1. Controlling Redundancy

**Redundancy:** storing the same data multiple times

- \* Duplication of effort
- \* Storage space is wasted
- \* Inconsistency

**Ideal situation:** each logical data item stored in only one place in the database- **data normalization**-ensures consistency and saves storage space.


**Controlled redundancy** - capability to control redundancy in order to prohibit inconsistencies among the files.





# Advantages of Using the Database Approach

## 2. Restricting Unauthorized Access

- Most users will not be authorized to access all information in the database. **Example:** financial data
  - Some users may only be permitted to retrieve data, whereas others are allowed to retrieve and update
  - A DBMS should provide a security and authorization subsystem, which the DBA uses to create accounts and to specify account restrictions
  - DBMS should enforce these restrictions automatically
  - Similar controls can be applied to the DBMS software- using certain privileged software
  - Parametric users may be allowed to access the database only through the predefined canned transactions developed for their use
- 



# Advantages of Using the Database Approach

## 3. Providing Persistent Storage for Program Objects

- Databases can be used to provide persistent storage for program objects and data structures
- Object-oriented database systems are compatible with programming languages such as C++ and Java, and the DBMS software automatically performs any necessary conversion



# Advantages of Using the Database Approach

## 4. Providing Storage Structures and Search Techniques for Efficient Query Processing

- Because the database is typically stored on disk, the DBMS must provide specialized data structures and search techniques to speed up disk search for the desired records
- Auxiliary files called **indexes** are used for this purpose
- In order to process the database records needed by a particular query, those records must be copied from disk to main memory
- **Query processing and optimization module** of the DBMS is responsible for choosing an efficient query execution plan for each query based on the existing storage structures



# Advantages of Using the Database Approach

## 5. Providing Backup and Recovery

Backup and recovery subsystem of the DBMS is responsible for hardware or software failures.

## 6. Enforcing Integrity Constraints

DBMS should provide capabilities for defining and enforcing these constraints

- Specifying a data type for each data item
- Referential integrity constraint
- Key or uniqueness constraint







# Advantages of Using the Database Approach


## 7. Representing Complex Relationships among Data

A database may include numerous varieties of data that are interrelated in many ways

A DBMS must have the capability to represent a variety of complex relationships among the data, to define new relationships as they arise, and to retrieve and update related data easily and efficiently

## 8. Providing Multiple User Interfaces

DBMS should provide a variety of user interfaces

- Query languages for casual users
  - Programming language interfaces for application programmers
  - Forms and command codes for parametric users,
  - Menu-driven interfaces and natural language interfaces for standalone user
- 



# Implications of Using the Database Approach


## 1. Potential for Enforcing Standards

- Database approach permits the DBA to define and enforce standards among database users in a large organization
- Facilitates communication and cooperation among various departments, projects, and users within the organization
- Can be defined for names and formats of data elements, display formats, report structures, terminology, and so on

**Examples:** Date data must be displayed in dd-mm-yyyy format

Names will be displayed as last name, first name and middle name

## 2. Reduced Application Development Time:

- Once a database is up and running, substantially less time is generally required to create new applications using DBMS facilities
- 

# Implications of Using the Database Approach

## 3. Flexibility

- It may be necessary to change the structure of a database as requirements change
- DBMSs allow certain types of evolutionary changes to the structure of the database without affecting the stored data and the existing application programs

## 4. Availability of Up-to-Date Information:

- As soon as one user's update is applied to the database, all other users can immediately see this update; **example: reservation systems**

## 5. Economies of Scale:

- Reduces the amount of wasteful overlap between activities of data-processing personnel in different projects or departments
- Organization can invest in more powerful processors, storage devices etc




## When not to use DBMS

**DBMS may involve unnecessary overhead costs due to:**

- High initial investment in hardware, software, and training
- The generality that a DBMS provides for defining and processing data
- Overhead for providing security, concurrency control, recovery, and integrity functions

**More desirable to use regular files under the following circumstances:**

- Simple, well-defined database applications that are not expected to change at all
  - Stringent, real-time requirements for some application programs that may not be met because of DBMS overhead
  - Embedded systems with limited storage capacity, where a general-purpose DBMS would not fit
  - No multiple-user access to data
- 



# Unit 1

# Database System Concepts And Architecture




# Data models, Schemas and Instances

## Data abstraction

- Suppression of details of **data organization and storage**, and the highlighting of the **essential features** for an improved understanding of data
- Different users can perceive data at their preferred level of detail

## Data model

- Provides the necessary means to achieve abstraction
  - A collection of concepts that can be used to describe the **structure** of a database
  - Structure of a database - data types, relationships, and constraints that apply to the data
  - Most data models also include a set of **basic operations** for specifying retrievals and updates on the database
- 




# Data models

## Categories of Data Models

### 1. High-level(conceptual, semantic) data models:

- Provide concepts that are close to the way many users perceive data
- Use concepts such as entities, attributes, and relationships, E.g: ER Model

### 2. Low-level or physical data models:


- Describe the details of how data is stored on the computer storage media
  - Generally meant for computer specialists, not for end users
  - Representing information such as record formats, record orderings, and access paths
- 



# Data models

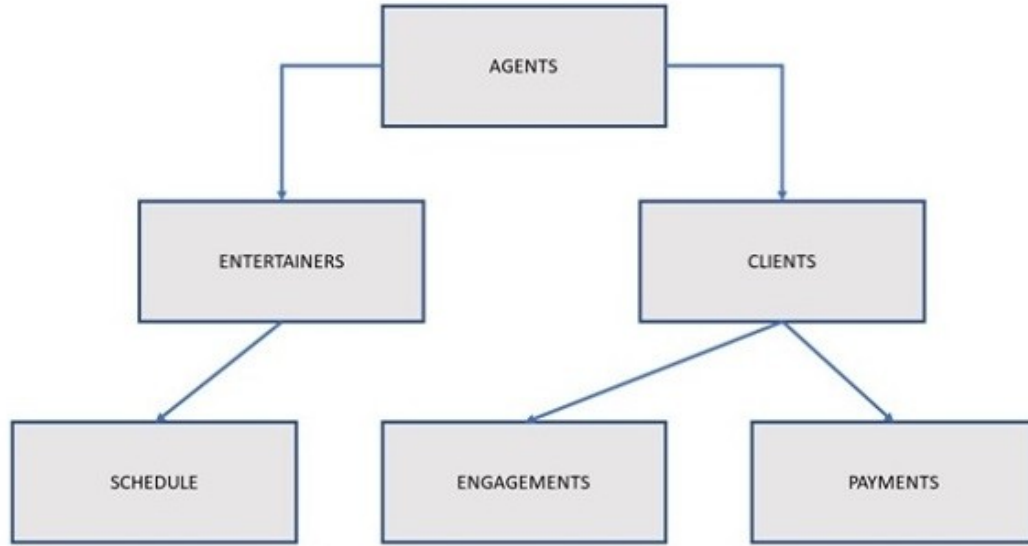
## Categories of Data Models

### 3. Representational (implementation, record-based) data models

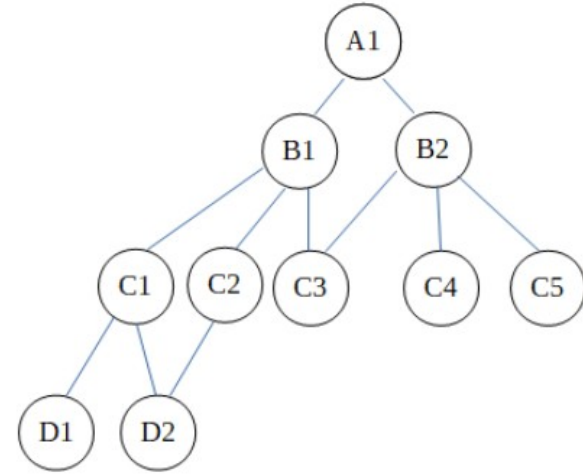
- Provide concepts that may be easily understood by end users but that are not too far removed from the way data is organized in computer storage
  - Hide many details of data storage on disk but can be implemented on a computer system directly
  - Used most frequently in traditional commercial DBMSs
  - e.g: relational data model, network and hierarchical models
- 



# Data models



A hierarchical model represents the data in a tree-like structure in which there is a single parent for each record.



In network data model, a child can be linked to multiple parents



# Schemas and Instances

## Database Schema:

- Description of a database
- Specified during database design and is not expected to change frequently
- Most data models have certain conventions for displaying schemas as diagrams
- A displayed schema is called a **schema diagram**
- A schema diagram displays only some aspects of a schema, such as the names of record types and data items, and some types of constraints
- **Schema construct**: each object in the schema



# Schemas and Instances

## STUDENT

Name	Student_number	Class	Major
------	----------------	-------	-------

## COURSE

Course_name	Course_number	Credit_hours	Department
-------------	---------------	--------------	------------

## PREREQUISITE

Course_number	Prerequisite_number
---------------	---------------------

## SECTION

Section_identifier	Course_number	Semester	Year	Instructor
--------------------	---------------	----------	------	------------


## GRADE\_REPORT

Student_number	Section_identifier	Grade
----------------	--------------------	-------



# Schemas and Instances


## Database State:

- The actual data in a database may change quite frequently
  - The data in the database at a particular moment in time is called a database state or snapshot
  - Also called the current set of occurrences or instances in the database
  - In a given database state, each schema construct has its own current set of instances
  - Every time we insert or delete a record or change the value of a data item in a record, we change one state of the database into another state
- 



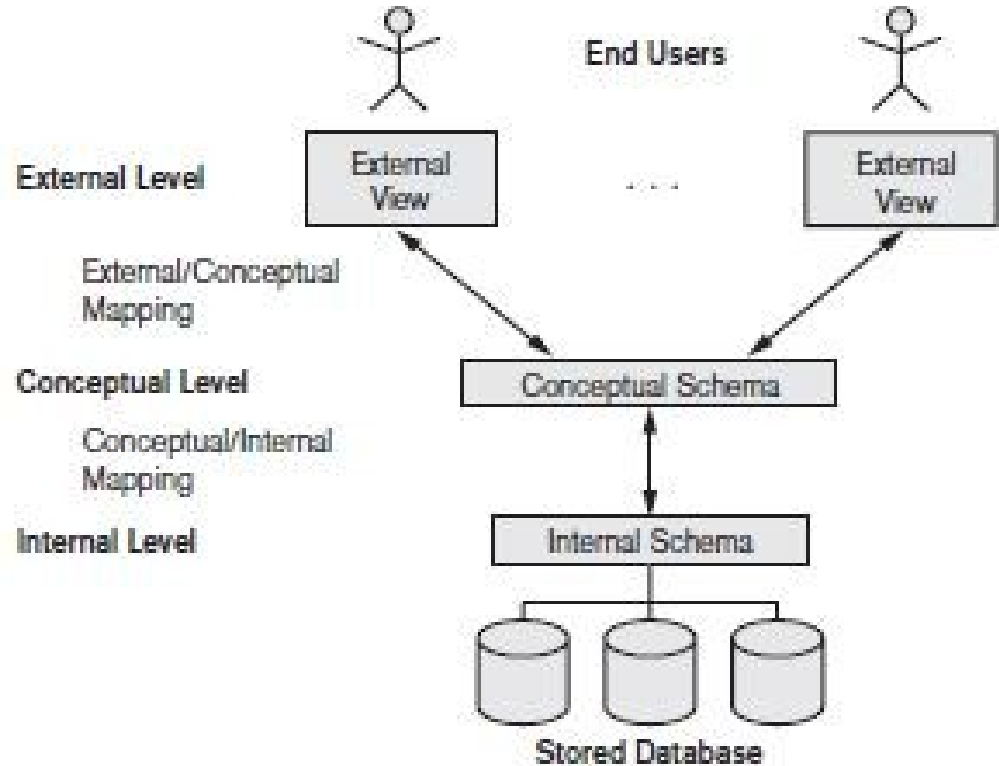
# Schemas and Instances

## Database State:

- When we define a new database, database state is the **empty state** with no data
  - We get the **initial state** of the database when the database is first populated with the initial data
  - At any point in time, the database has a current state
  - **A valid state**—a state that satisfies the structure and constraints specified in the schema
  - The schema is also called the **intension**, and a database state is called an **extension** of the schema
- 

# The Three-Schema Architecture

- Proposed to help achieve and visualize characteristics of the database approach (self-describing, program-data and program-operation independence, support of multiple user views)
- Goal is to separate the user applications from the physical database





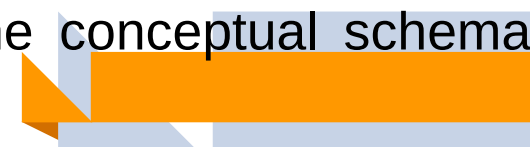
# The Three-Schema Architecture

Schemas can be defined at the following three levels :

## 1. The internal level has an internal schema

- Describes the physical storage structure of the database
- 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

- Describes the structure of the whole database for a community of users
  - 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
- 



# The Three-Schema Architecture


## 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
  - Typically implemented using a representational data model
  - Most DBMSs do not separate the three levels completely and explicitly, but support the three-schema architecture to some extent
- 





# The Three-Schema Architecture

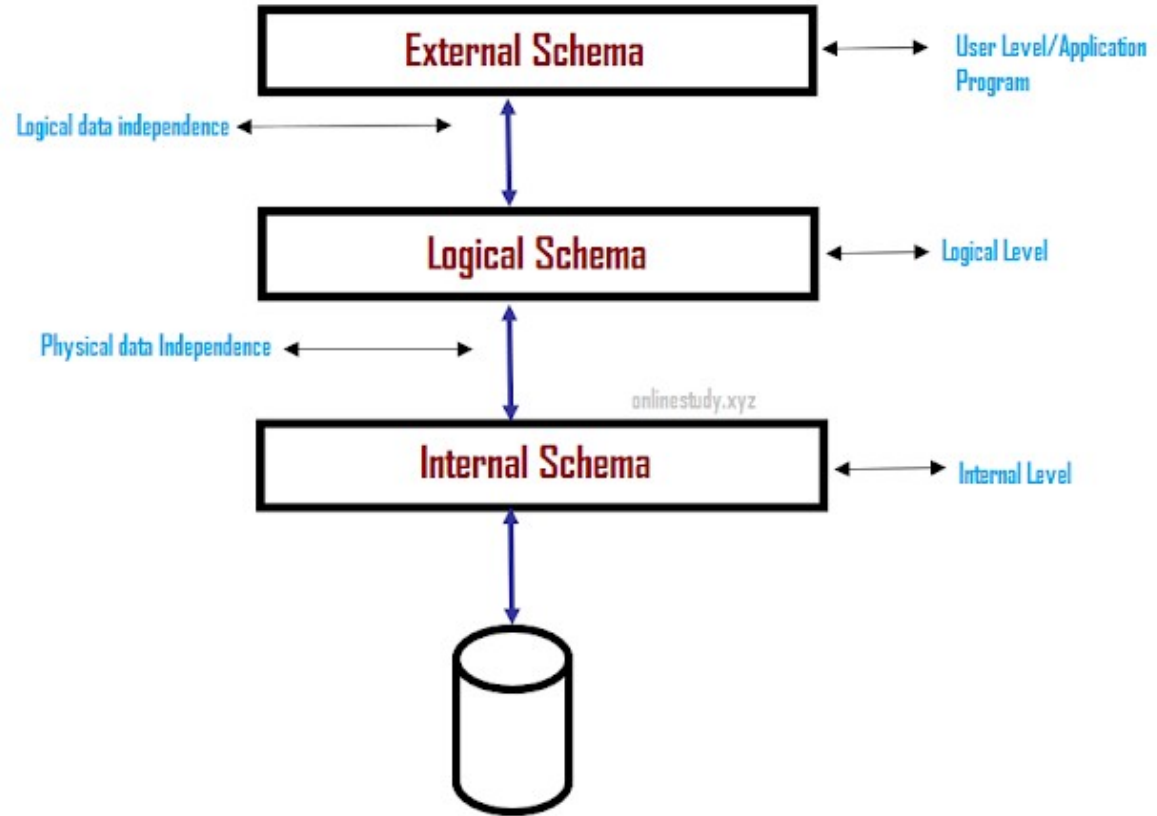
- In a DBMS based on the three-schema architecture, each user group refers to its own external schema.
  - Hence, the DBMS must transform a request specified on an external schema into a request against the **conceptual schema**, and then into a request on the **internal schema** for **processing** over 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

Capacity to change the schema at one level of a database system without having to change the schema at the next higher level


Types:

1. Logical Data Independence
2. Physical Data Independence





# Logical Data Independence

- 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)
  - Only the view definition and the mappings need to be changed in a DBMS that supports logical data independence
  - After the conceptual schema undergoes a logical reorganization, application programs that reference the external schema constructs must work as before
  - Changes to constraints can be applied to the conceptual schema without affecting the external schemas or application programs
- 



# Physical Data Independence

- Capacity to change the internal schema without having to change the conceptual schema
- External schemas need not be changed as well
- Changes to the internal schema may be needed because some physical files were reorganized
- e.g: creating additional access structures—to improve the performance of retrieval or update

***When a schema at a lower level is changed, only the mappings between this schema and higher level schemas need to be changed***

***The higher-level schemas themselves are unchanged.***






# **Unit 1**

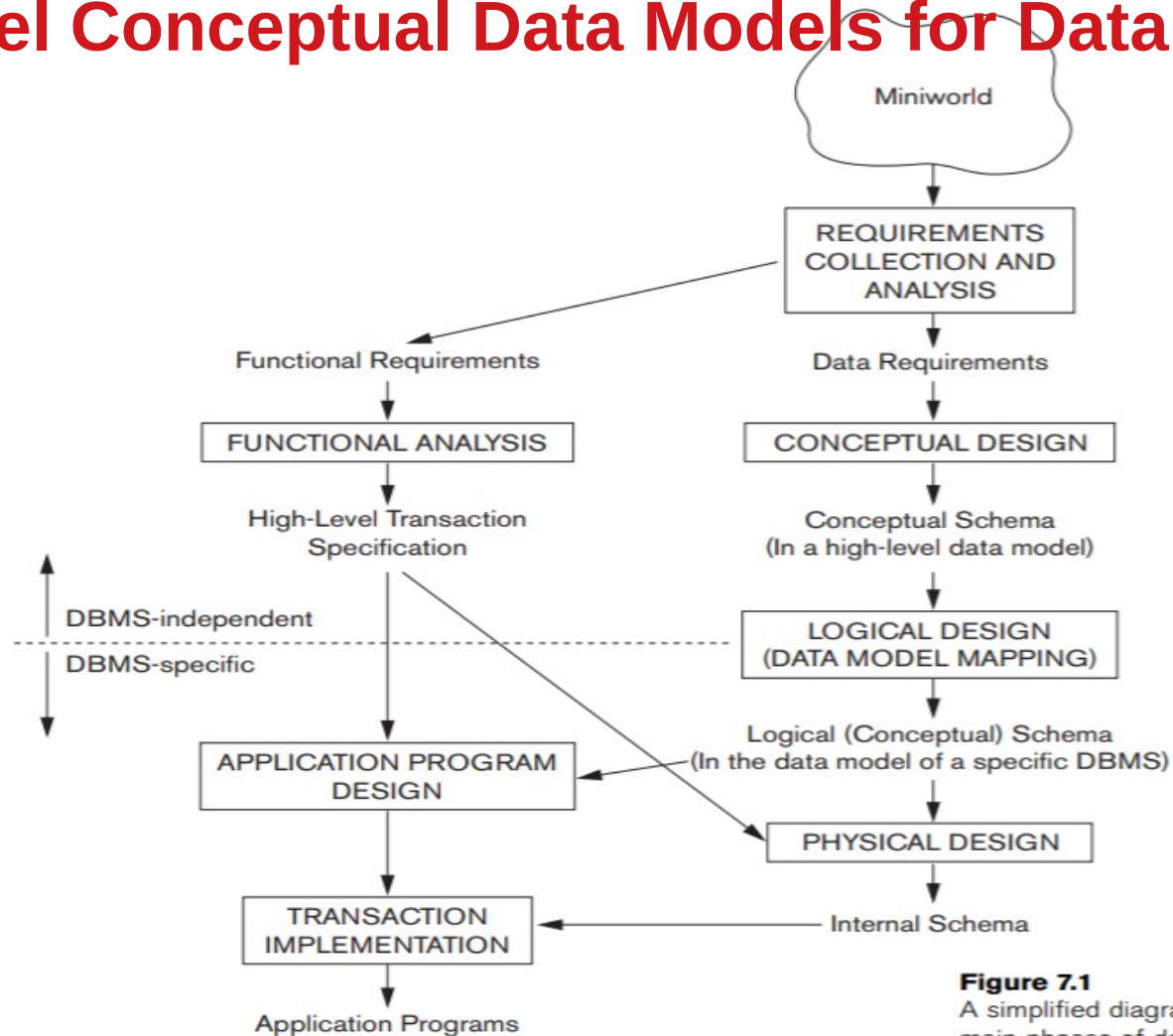
# **Entity-Relationship Model**



# ER Model

- ER model stands for an **Entity-Relationship model**.
  - It is a **high-level** data model. This model is used to define the data elements and relationship for a specified system.
  - It develops a **conceptual design** for the database. It also develops a very simple and easy to design view of data.
  - In ER modeling, the database structure is portrayed as a diagram called an **entity-relationship diagram**.
  - It works around real-world entities and the associations among them. At view level, the ER model is considered a good option for designing databases.
- 

# High-Level Conceptual Data Models for Database Design




**Figure 7.1**

A simplified diagram to illustrate the main phases of database design.



# Database Design Process - Steps

## 1. Requirements Collection and Analysis

- 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 and complete a form as possible
  - 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
  - **Data flow diagrams, Sequence diagrams, Scenarios**, and other techniques are used to represent functional requirements
- 





# Database DesignProcess - Steps

## 2. Conceptual design


- Create a conceptual schema for the database, using a high-level conceptual data model
- Conceptual schema Concise description of the data requirements of the users
- Includes detailed descriptions of the entity types, relationships, and constraints
- Expressed using the concepts provided by the high-level data model
- Easier to understand and can be used to communicate with nontechnical users
- Can be used as a reference to ensure that all users' data requirements are met and that the requirements do not conflict





# Database DesignProcess - Steps


## 2. Conceptual design (Continued)

- 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 to the conceptual schema can be introduced if some functional requirements cannot be specified using the initial schema
- 



# Database DesignProcess - Steps

## 3. Logical design or data model mapping

- Actual implementation of the database, using a commercial DBMS
  - Most current commercial DBMSs use an implementation data model—such as the **relational or the object-relational database model**
  - Conceptual schema is transformed from the high-level data model into the implementation data model
  - Result is a database schema in the implementation data model of the DBMS
  - Data model mapping is often automated or semiautomated within the database design tools
- 



# Database DesignProcess - Steps

## 4. Physical design

- 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 as database transactions corresponding to the high level transaction specifications

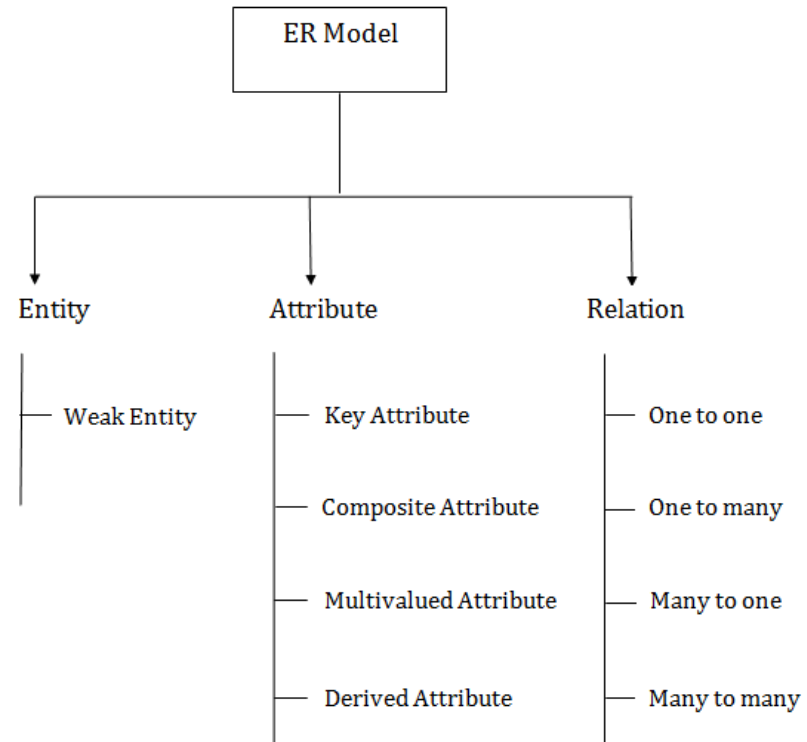


# Introduction to Entity-Relationship (E-R) Modeling

- The ER model describes data as **entities**, **relationships**, and **attributes**
- A detailed, logical representation of the entities, associations and data elements for an organization or business is called **Entity-Relationship (E-R) Diagram**

## Key Terms

- Entity
- Entity Types
- Entity Sets
- Attributes
- Relationship
- Relationship Types
- Relationship Sets
- Structural Constraints



# Components of the ER Diagram

This model is based on three basic concepts: **Entities** , **Attributes**, **Relationships**



Entity Name

**Entity**

Person, place, object, event  
or concept about which  
data is to be maintained

**Example:** Car, Student



Jack



Attribute  
Name

**Attribute**

Property or characteristic of  
an entity

**Example:** Color of car Entity  
Name of Student Entity



**Relation**



Verb  
Phrase

Association between the instances of one or  
more entity types

**Example:** Blue Car Belongs to Student Jack



# Entity

A thing in the real world with an independent existence

May be an object with a Physical existence

**Example**, a particular person, car, house, or employee)

Or it may be an object with a Conceptual existence

Example: a job, or a university course

Real-world object distinguishable from other objects

*(e.g a student, car, job, subject, building ...)*

Each entity has attributes—the particular properties that describe it

**Example:** employee's name, age, address, salary, and job

A particular entity will have a value for each of its attributes





# Entities and Attributes

- Each entity has **Attributes**—the particular properties that describe it
- In ER diagrams place attributes name in an **ellipse** with a line connecting it to its associated entity
- Attributes may also be associated with relationships
- An attribute is associated with exactly **one entity** or **relationship**
- The same entity may have different **prominence** in different UoDs (miniworld or the Universe of Discourse)

In the Company database, an employee's **car** is of lesser importance

In the Department of Transportation's registration database, **cars** may be the most important concept

In both cases, **cars** will be represented as entities; but with different levels of detail








# Entities and Attributes

## Types of attributes

### Simple or Atomic attributes

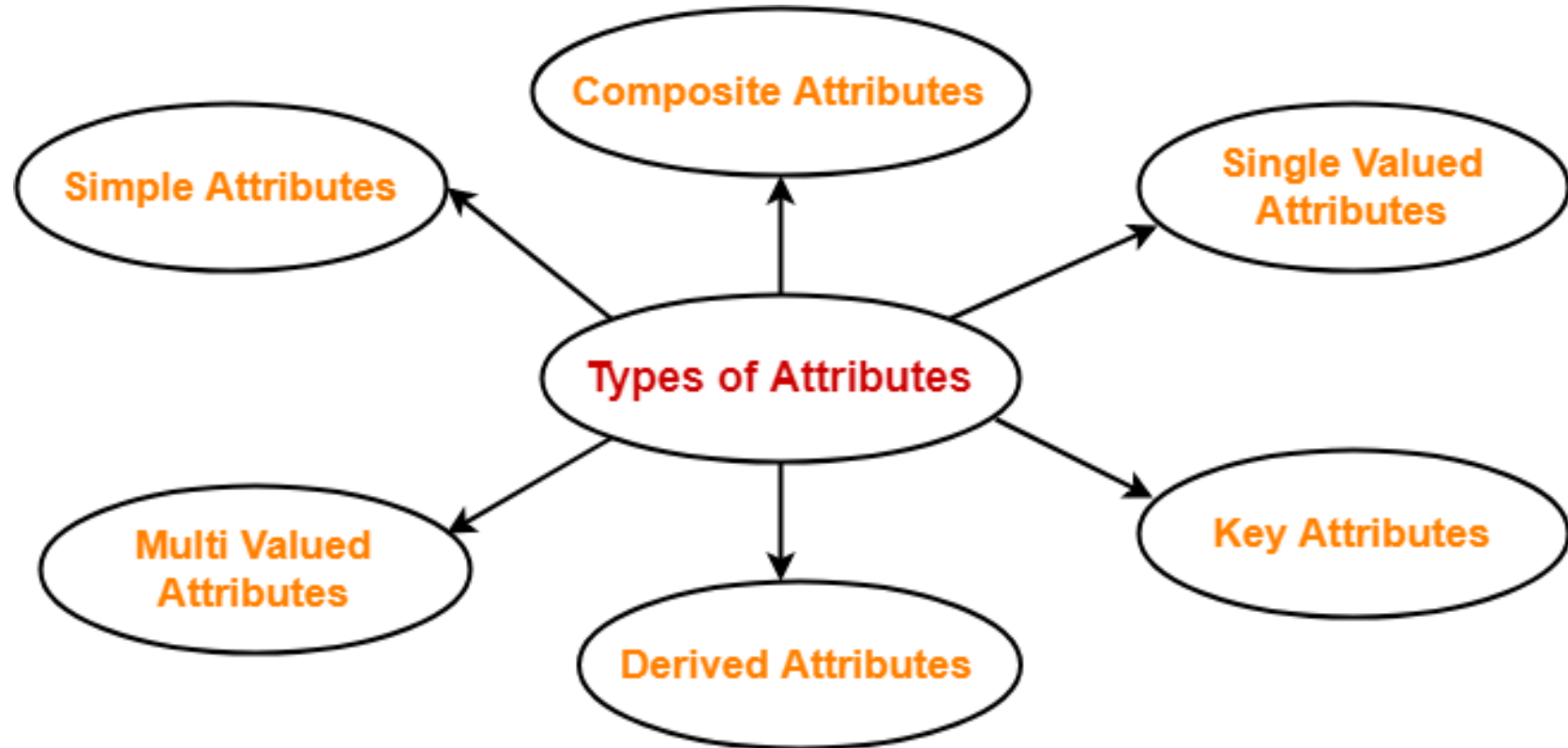
- Each entity has a single atomic value for the attribute
- Attributes that are not divisible

### Composite attributes

- Can be divided into smaller subparts, which represent more basic attributes with independent meanings
  - Can form a hierarchy
  - 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
- 

# Entities and Attributes

## Types of attributes






# Entities and Attributes

## Types of attributes

### Simple or Atomic attributes

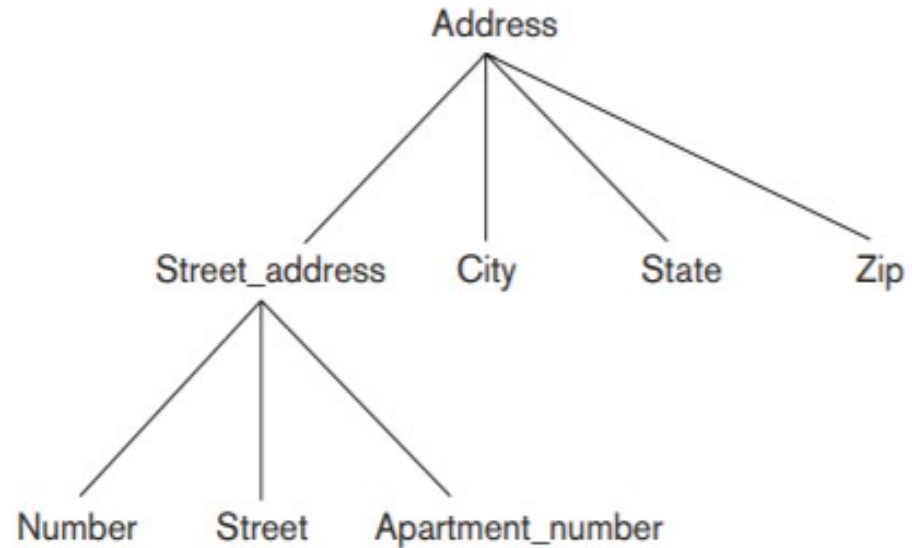
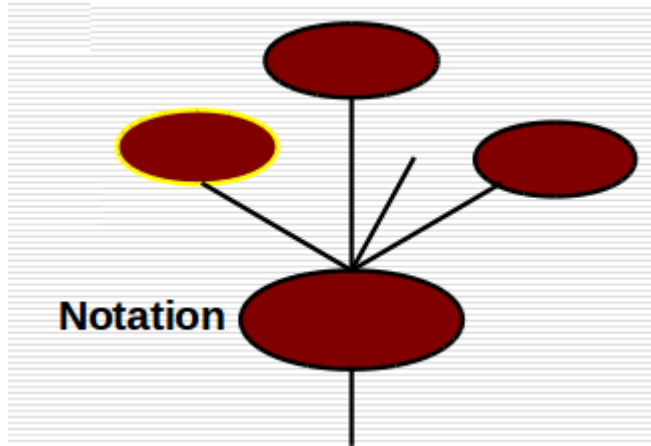
- Each entity has a single atomic value for the attribute
- Attributes that are not divisible

### Composite attributes

- Can be divided into smaller subparts, which represent more basic attributes with independent meanings
  - Can form a hierarchy
  - 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
- 

# Entities and Attributes

## Types of attributes ( Simple vs Composite attributes)



**Attribute Domain: The set of allowable values for one or more attributes.**

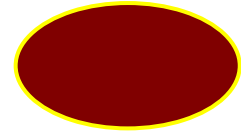
# Entities and Attributes

## Notation

### Types of attributes

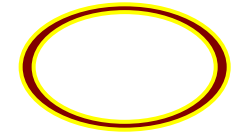
#### Single-Valued

- Attributes which have a single value for a particular entity
- Gender = F



#### Multi valued

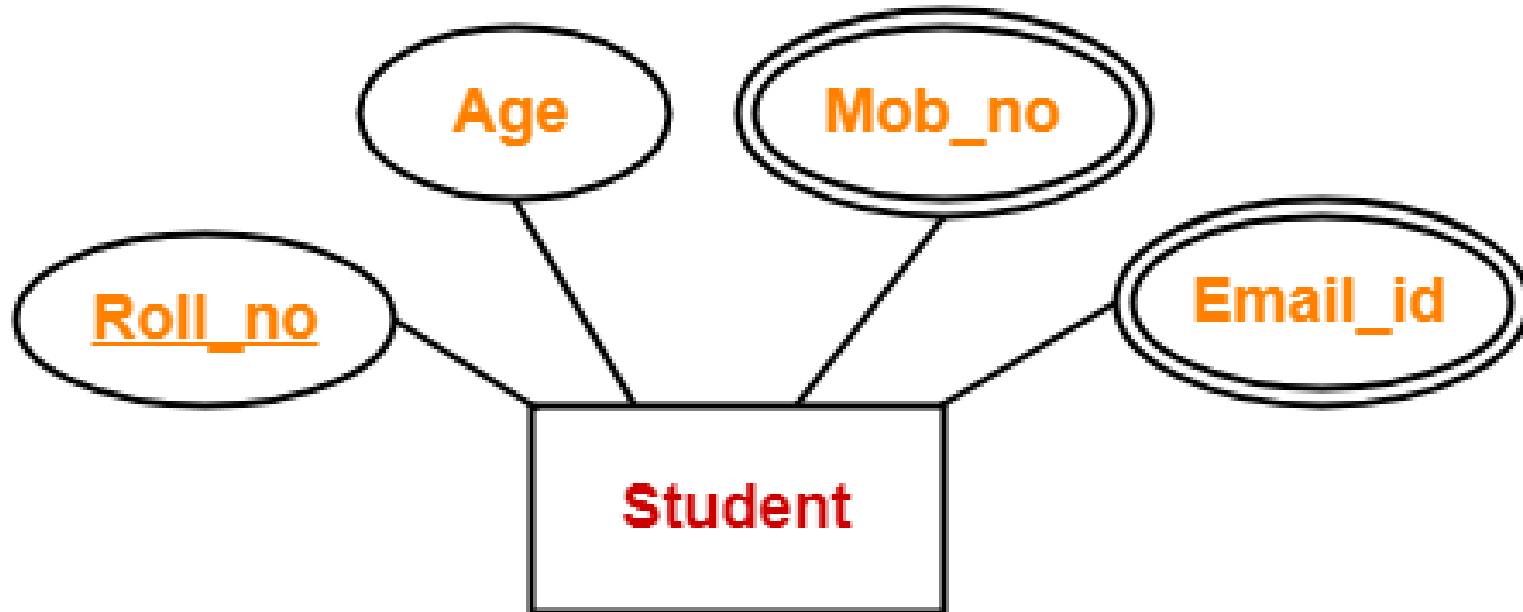
- An entity may have multiple values for that attribute
- A multivalued attribute may have lower and upper bounds to constrain the number of values allowed for each individual entity
- Degree = {BSc, MCA, Ph.D}



***... An “attribute” in the relational model is always single valued - Values are atomic!***

# Entities and Attributes

## Types of attributes (Single-Valued vs Multivalued)



# Entities and Attributes

## Types of attributes

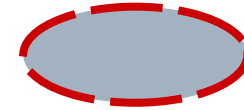
### Stored attribute

- The value of an attribute is stored

### Derived attribute

- The value of an attribute can be determined from the value of another attribute
- **Example:** Age from DOB, Duration from starting time and ending time, etc

### Notation

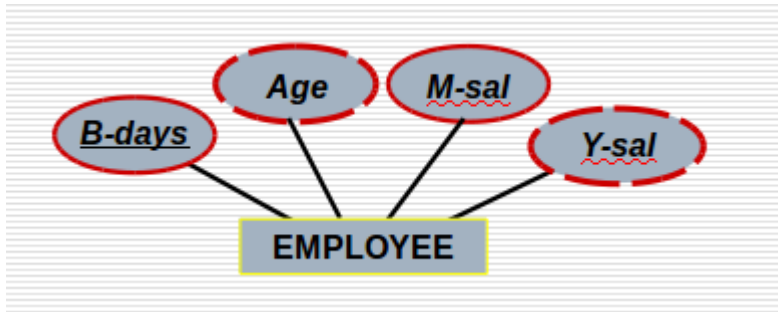


# Entities and Attributes

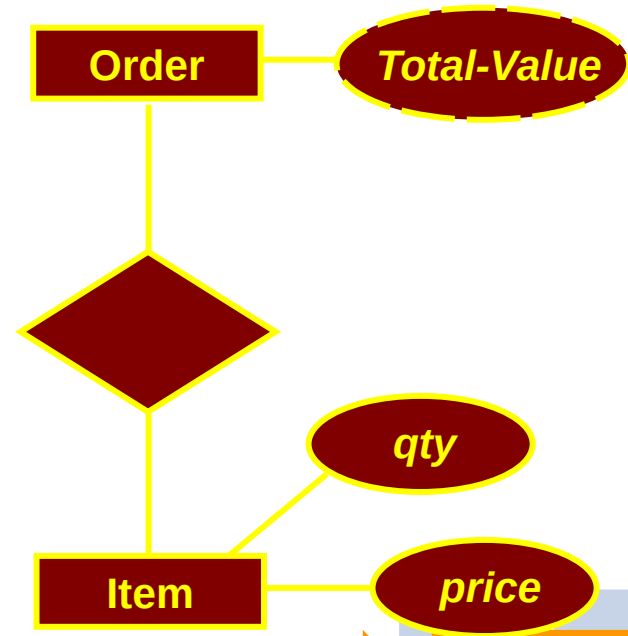
## Types of attributes ( Stored Vs Derived )

Age ® Date - B-day

Y-Sal ®  $12 * M\text{-Sal}$



Total-value ®  $\text{sum}(\text{qty} * \text{price})$



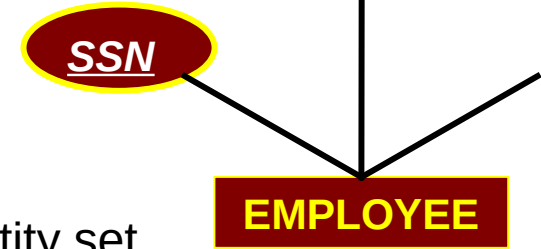


# Entities and Attributes

## Types of attributes - Key attributes

### 1. Simple Key Attributes (Identifier attribute)

- Identifier attribute or Key is an attribute (or combination of attributes) that uniquely identifies individual instances of an entity type
- Each entity instance must have a single value for the attribute, and the attribute must be associated with each entity
- Key (or uniqueness) constraints are applied to entity types
- Key attribute's values are distinct for each individual entity in the entity set
- A key attribute has its name underlined inside the oval
- Key must hold for **every** possible extension of the entity type
- Multiple keys are possible ( Candidate Keys)

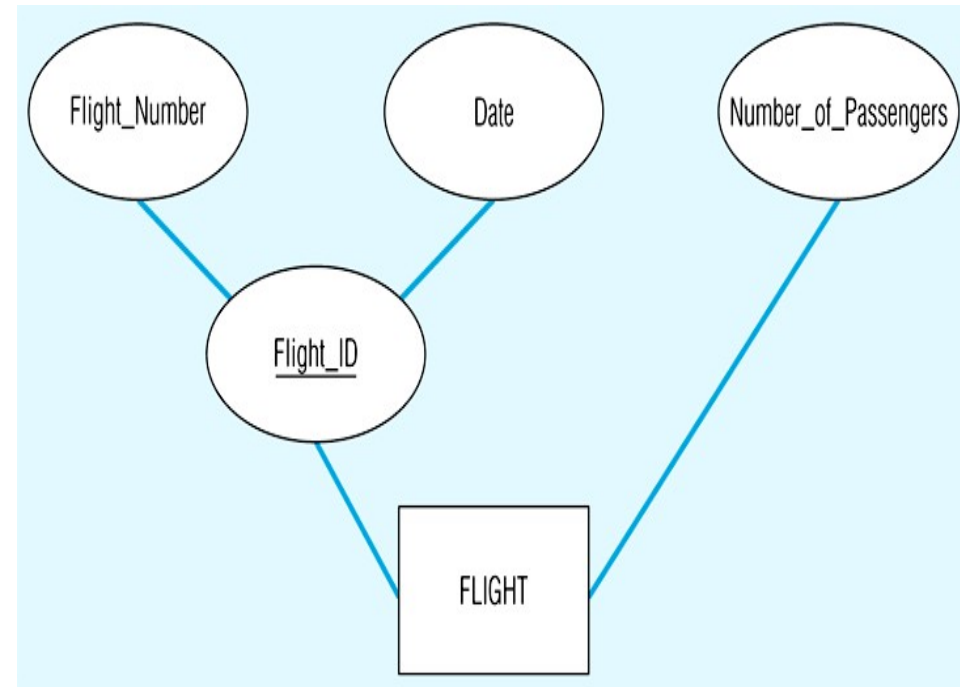


# Entities and Attributes

## Types of attributes - Key attributes

### 2. Composite Key Attributes

- A Composite Identifier is when there is no single (or atomic) that can serve as an identifier
- Flight\_ID is a composite identifier that has component attributes Flight\_Number and Date – this combination is required to uniquely identify individual occurrences of Flight
- Flight\_ID is underlined, whilst its components are not






# Entities and Attributes

## Types of attributes

### Null Valued Attributes

- An attribute, which has not any value for an entity is known as null valued attribute.
  - Tertiary-Degree ( third-level) : Not applicable for a person with no university education
- Home-Phone: Not known if it exists  
Height: Not known at present time

### Type of Null Values

- Not Applicable
  - Unknown
  - Missing
- 

# Entities and Attributes

## Types of attributes

### 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 { }

Parenthesis ( ) for composite attributes

Brackets { } for multi-valued attributes

Assume a person can have more than one residence and each residence can have multiple telephones

{AddressPhone

{ Phone ( AreaCode,PhoneNum ) },

Address (StreetAddresss (Number, Street, AptNo), City,State,PostalCode) ) }

## 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
- Similar to the basic data types available in most programming languages (e.g:integer, string, Boolean, float etc)
- Not typically displayed in basic ER diagrams
- **Examples:** Value set of the Age attribute of EMPLOYEE to be the set of integer numbers between 21 and 55, Gender attribute of an employee may have M or F or T.

Rooms in hotel (1-300)

Age (1-99)

Married (yes or no)

Nationality (Nepalese, Indian, American, or British)

Colors (Red, Yellow, Green)

- **Examples:**

# Entity Types

- Collection (or set) of entities that have the same attributes
- **E.g:** A company employing hundreds of employees may want to store similar information concerning each of the employees
- Each entity has its own value(s) for each attribute

- **Notation :**



Entity Type Name:

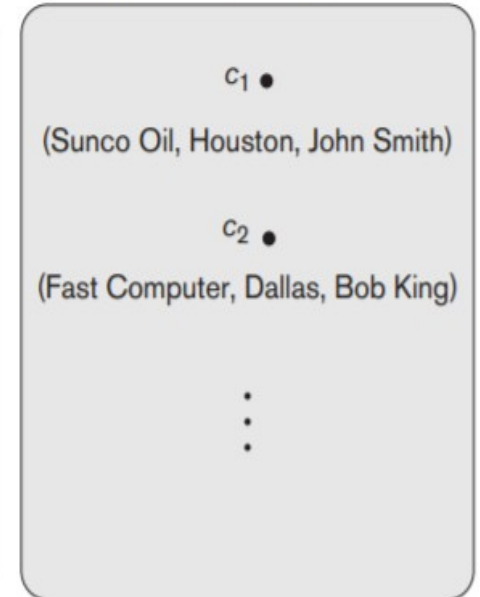
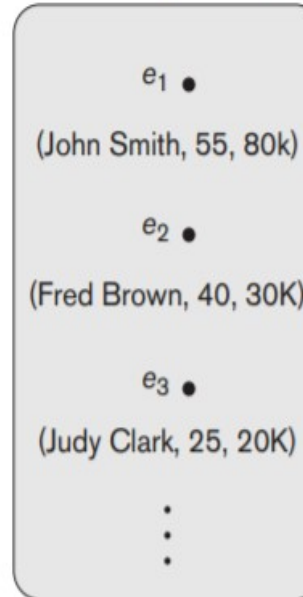
EMPLOYEE

COMPANY

Name, Age, Salary

Name, Headquarters, President

Entity Set:  
(Extension)





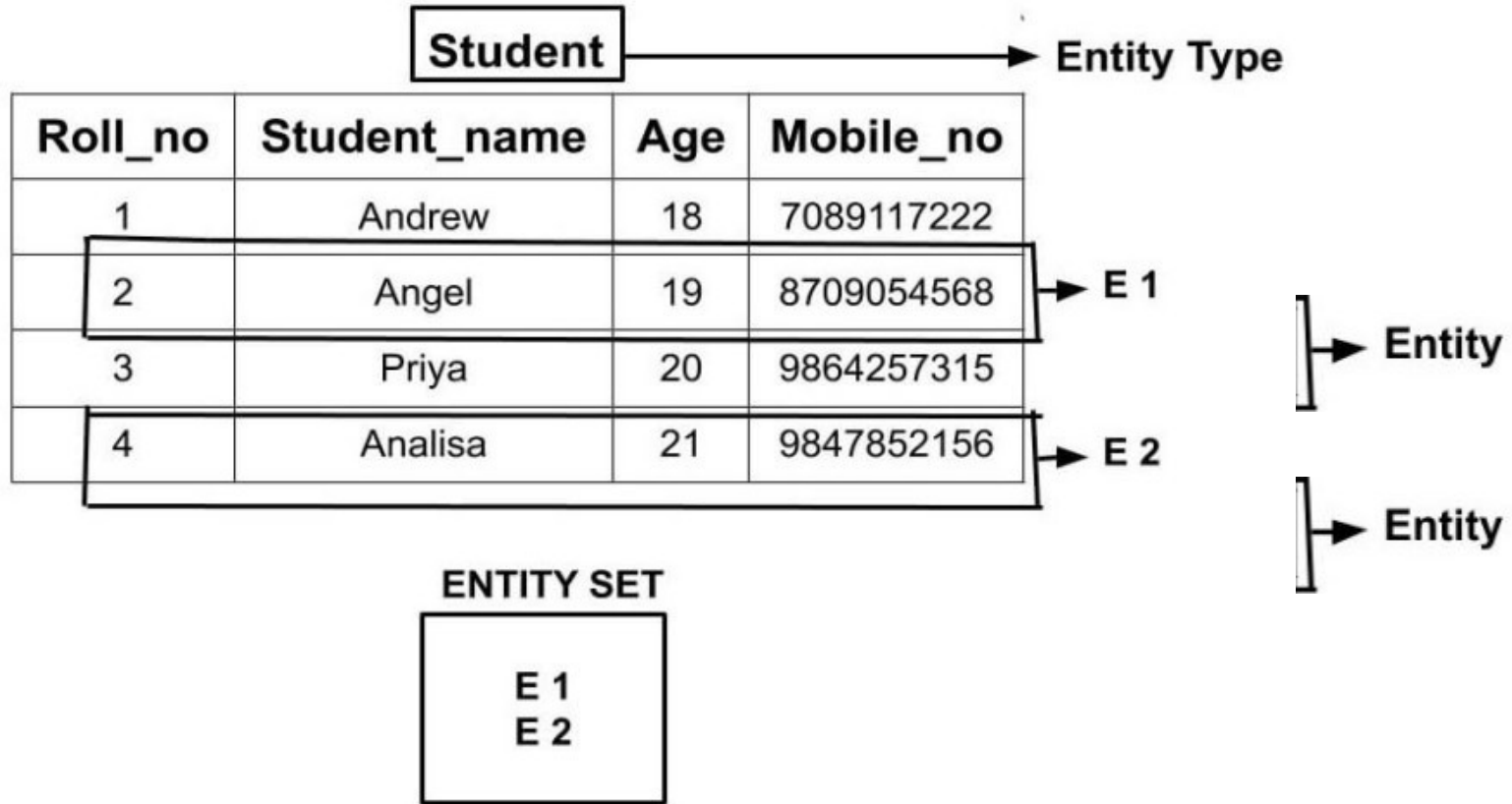
## Entity Sets or Entity collection

- The collection of all entities of a particular entity type in the database at any point in time or Current state of the entities of that type that are stored in the database.
- Usually referred to using the same name as the entity type
- **E.g:** EMPLOYEE refers to both a type of entity as well as the current collection of all employee entities in the database
- 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

*Entity Type and Entity Set are customarily referred to by the same name*



## Entity, Entity Types and Entity Sets

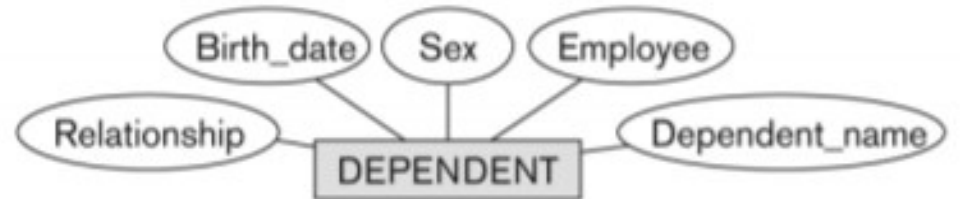
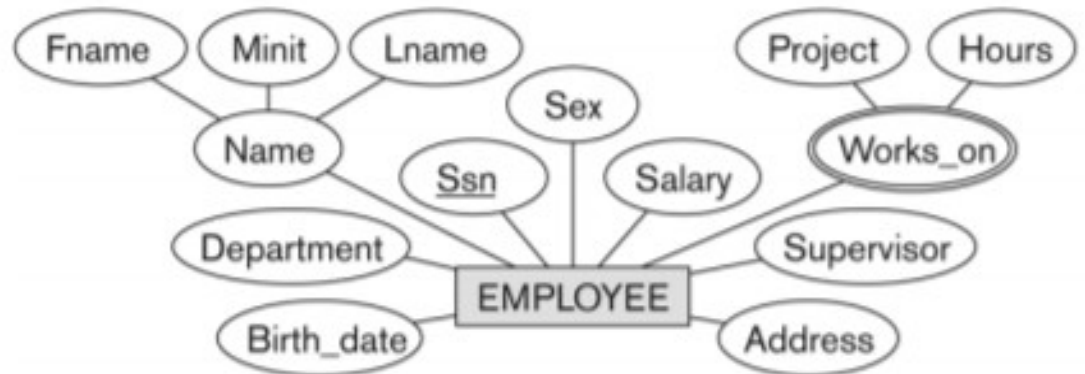
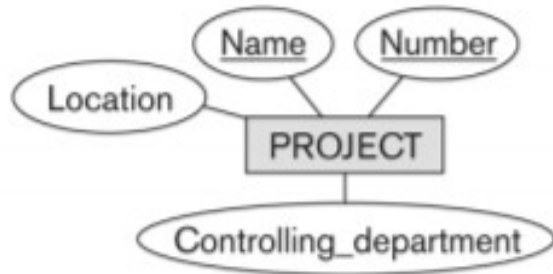
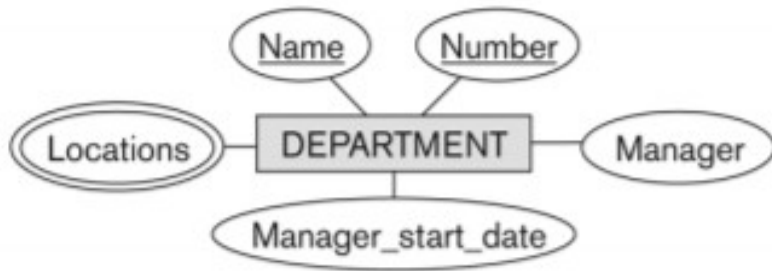


*Entity type is a superset of the entity set as all the entities are included in the entity type.*



## Design of Entity Types:

### EMPLOYEE, DEPARTMENT, PROJECT, DEPENDENT






## Relationships

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

*A Relationship is an association among two or more entities*

- E.g.: Student Ramesh enrolls in Discrete Mathematics course
  - • Relationship enrolls has Student and Course as the participating entities.
  - • Formally, enrolls  $\subseteq$  Student  $\times$  Course
  - •  $(s,c) \in \text{enrolls} \Leftrightarrow$  Student 's' has enrolled in Course 'c'
  - E.g: Department of EMPLOYEE refers to the department for which the employee works
- 



## Relationship Types

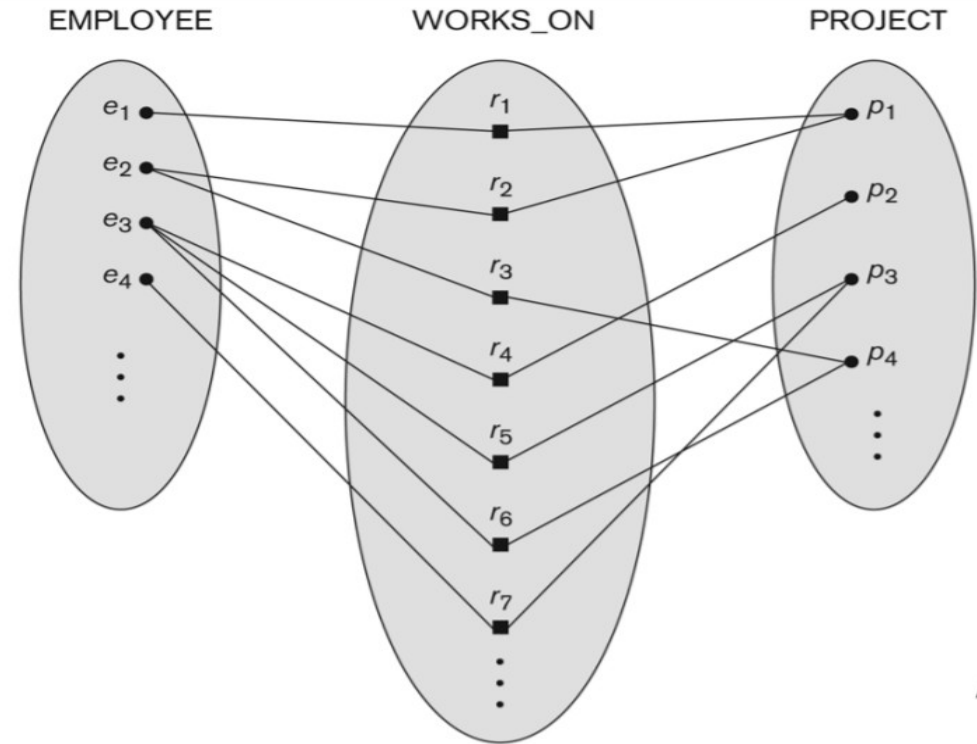
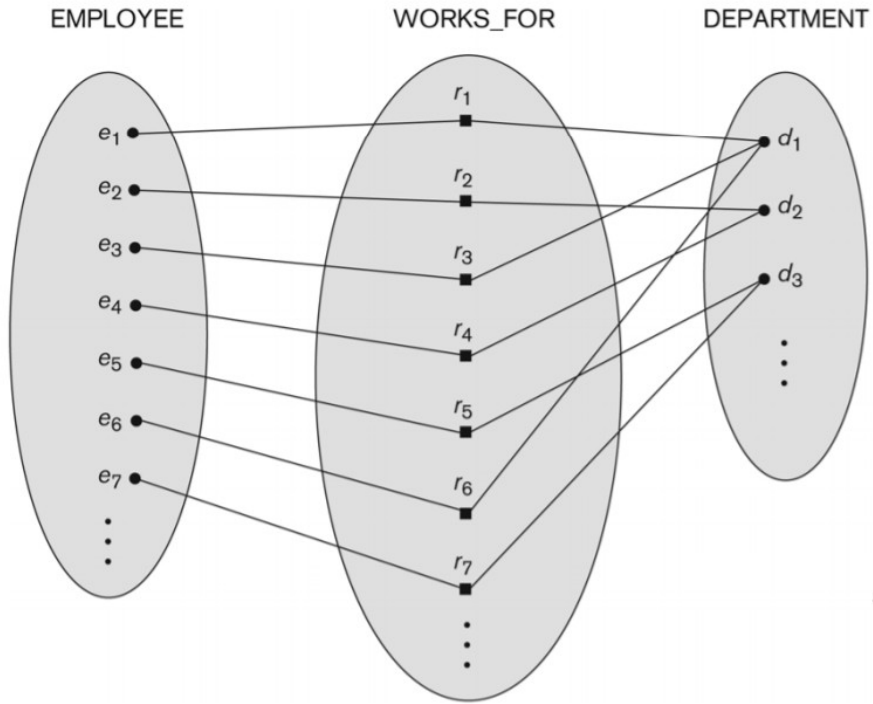
- Relationships of the same type
- A relationship type  $R$  among  $n$  entity types  $E_1, E_2, \dots, E_n$  defines a set of associations ( relationship set) among entities from these entity types
- E.g : 1. enrolls is called a relationship Type/Set.( in the previous example)
- 2. works\_for is called a relationship Type/Set between entity types Employee and Department



## Relationship Sets

- The current state of a relationship type
- Set of relationship instances  $r_i$ , where each  $r_i$  associates  $n$  individual entities ( $e_1, e_2, \dots, e_n$ )
- Each entity  $e_j$  in  $r_i$  is a member of entity set  $E_j$
- Relationship set is a mathematical relation on  $E_1, E_2, \dots, E_n$
- Can also be defined as a subset of the Cartesian product of the entity sets  $E_1 \times E_2 \times \dots \times E_n$
- Each of the entity types  $E_1, E_2, \dots, E_n$  is said to participate in the relationship type  $R$
- Each of the individual entities  $e_1, e_2, \dots, e_n$  is said to participate in the relationship instance  $r_i = (e_1, e_2, \dots, e_n)$

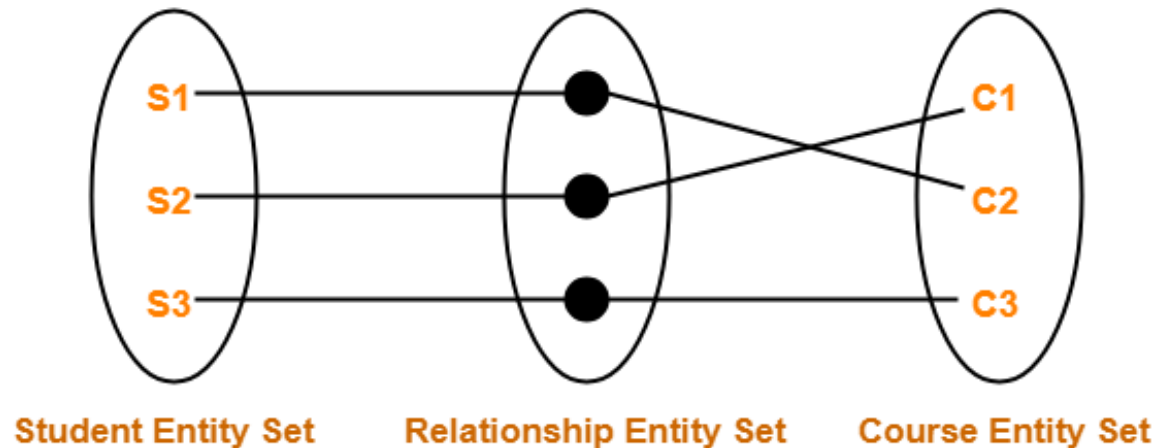
# Relationship, Relationship Types and Sets



# Relationship, Relationship Types and Sets

## Diagrammatic Notation for Relationships

- Relationship – diamond shaped box
- Rectangle of each participating entity is connected by a line to this diamond. Name of the relationship is written in the box.

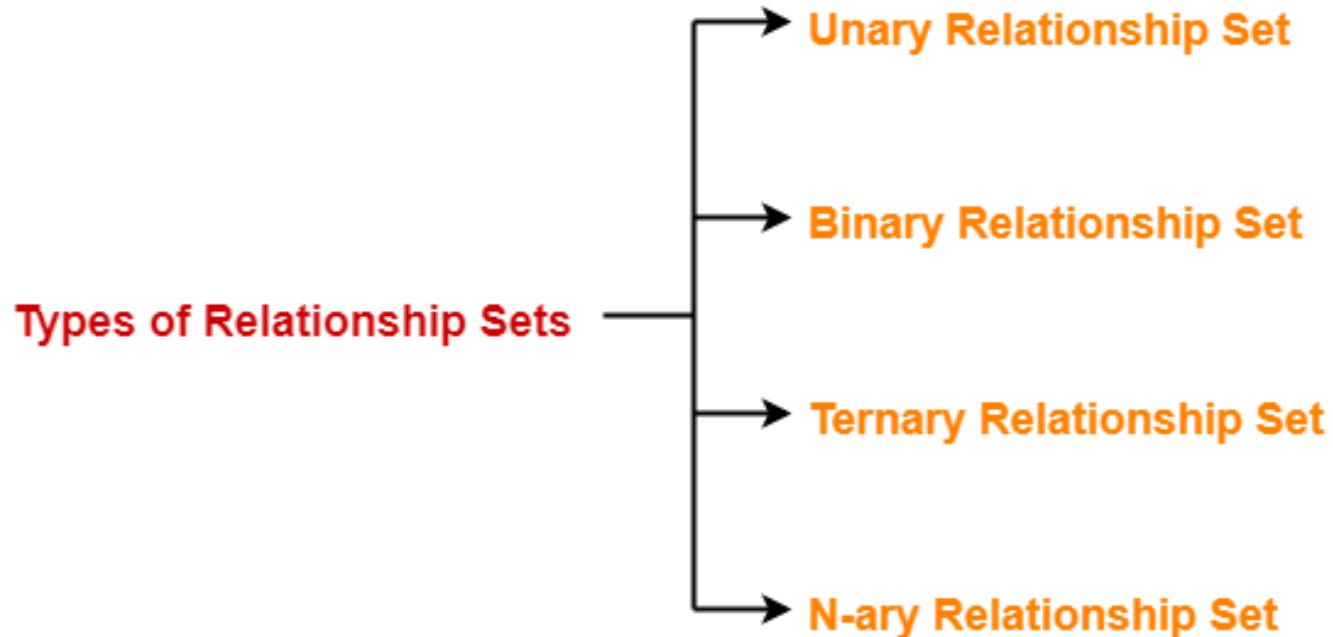


Set Representation of ER Diagram

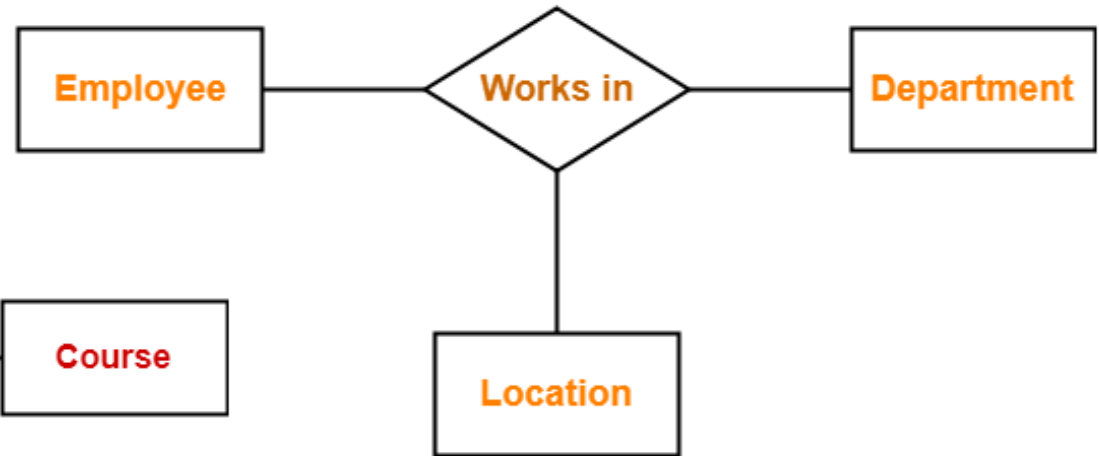
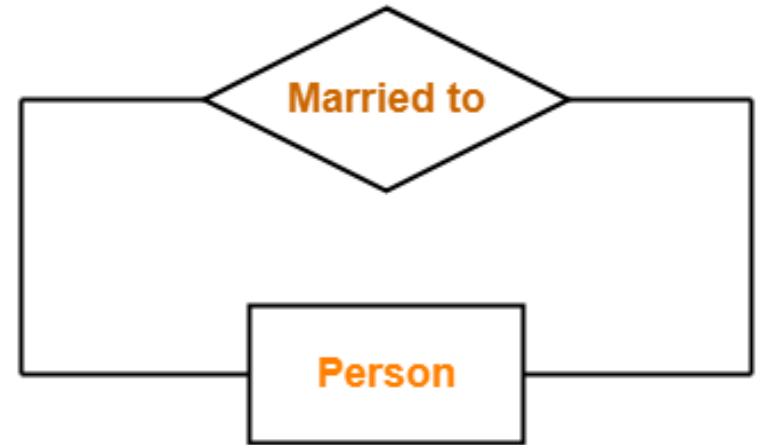
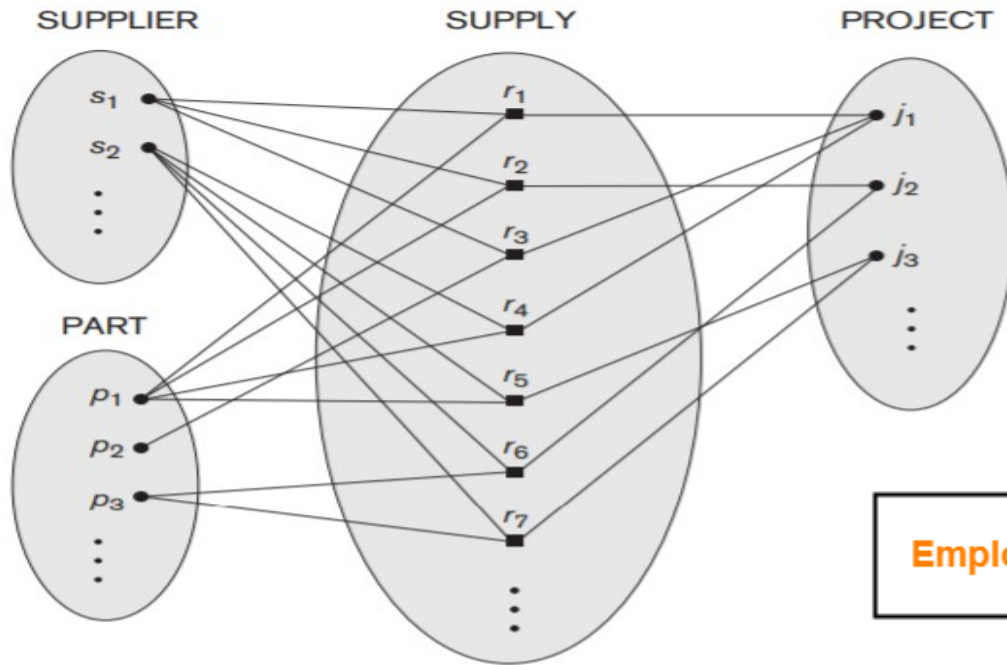
## Degree of a Relationship Set-

Degree of a relationship set = Number of entity sets participating in a relationship set

On the basis of degree of a relationship set, a relationship set can be classified into the following types-



## Degree of a Relationship Set- Examples








## Role Names


- Each entity type that participates in a relationship type plays a particular role in the relationship.
- Role name signifies the role that a participating entity from the entity type plays in each relationship instance.
- Helps to explain what the relationship means.
- 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.

E.g: in the WORKS\_FOR relationship type, EMPLOYEE plays the role of **employee or worker** and DEPARTMENT plays the role of **department or employer**



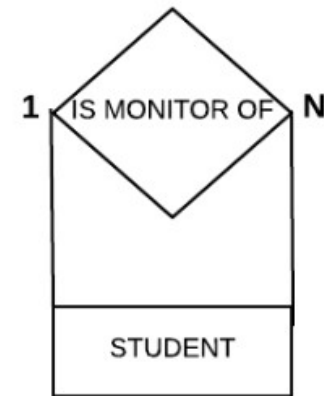
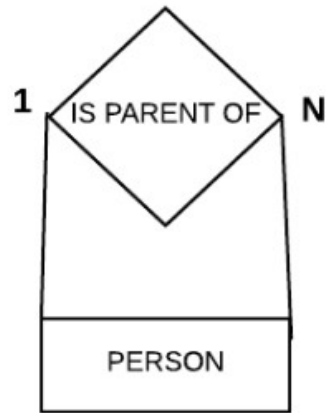
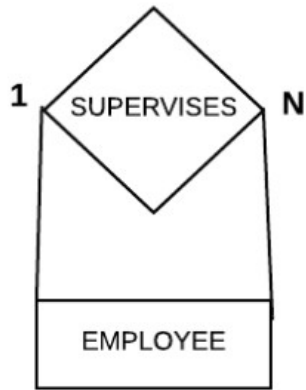


## **Recursive relationships or self-referencing relationships**

- Relationships where the same entity type participates more than once in a relationship type in different roles.
  - Or When there is a relationship between two entities of the same type, it is known as a recursive relationship. This means that the relationship is between different instances of the same entity type.
  - Role names are essential for distinguishing the meaning of the role that each participating entity plays
- 

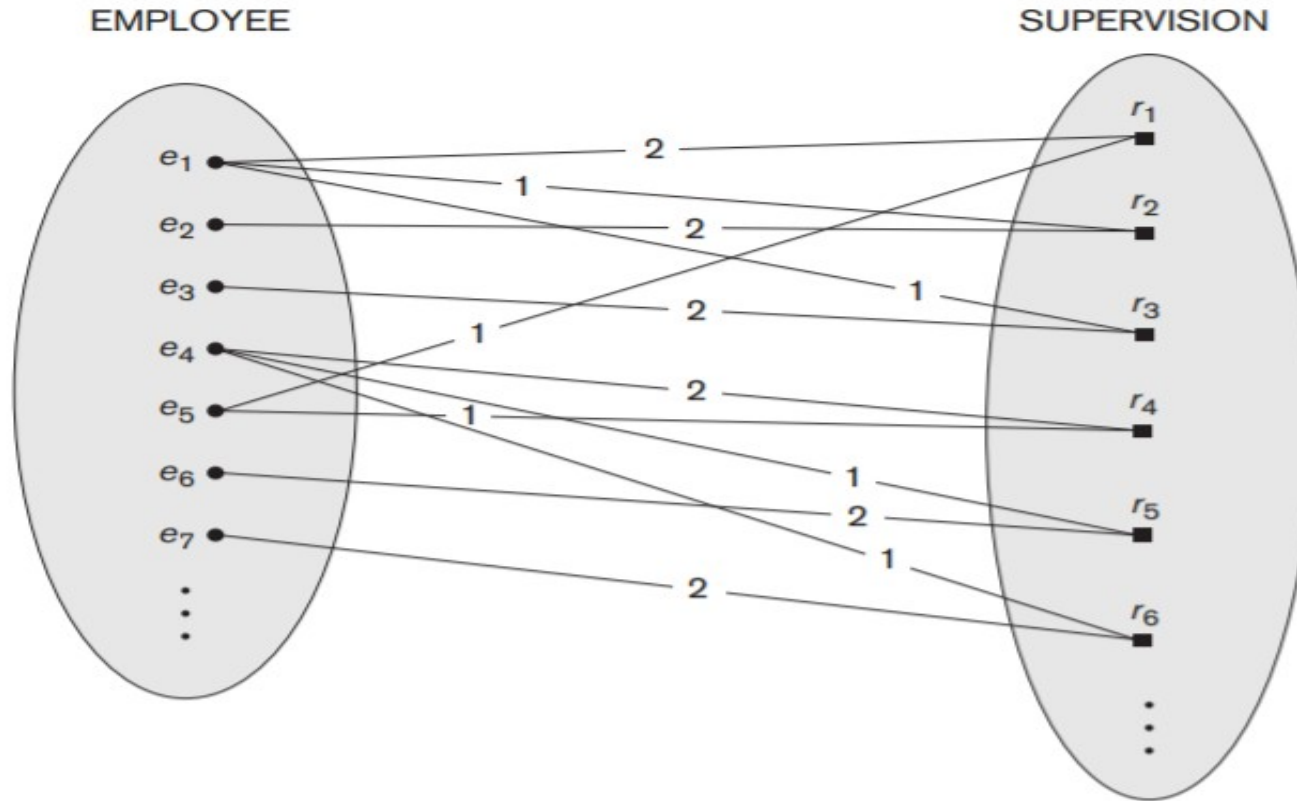
## Recursive relationships or self-referencing relationships

Some examples of recursive relationship



An employee can supervise multiple employees, A person can have many children who are also persons, A student can be a class monitor and handle other students. Hence, this is a recursive relationship of entity employee with itself, entity person with itself and entity student with itself . All these have 1 to many recursive relationship.

# Recursive relationships or self-referencing relationships






## Constraints on Binary Relationship Types

- Relationship types usually have certain constraints that limit the possible combinations of entities that may participate in the corresponding relationship set.
- These constraints are determined from the mini-world situation that the relationships represent

Example: Company's rule that each employee must work for exactly one department

### Types of binary relationship constraints(Structural constraints)

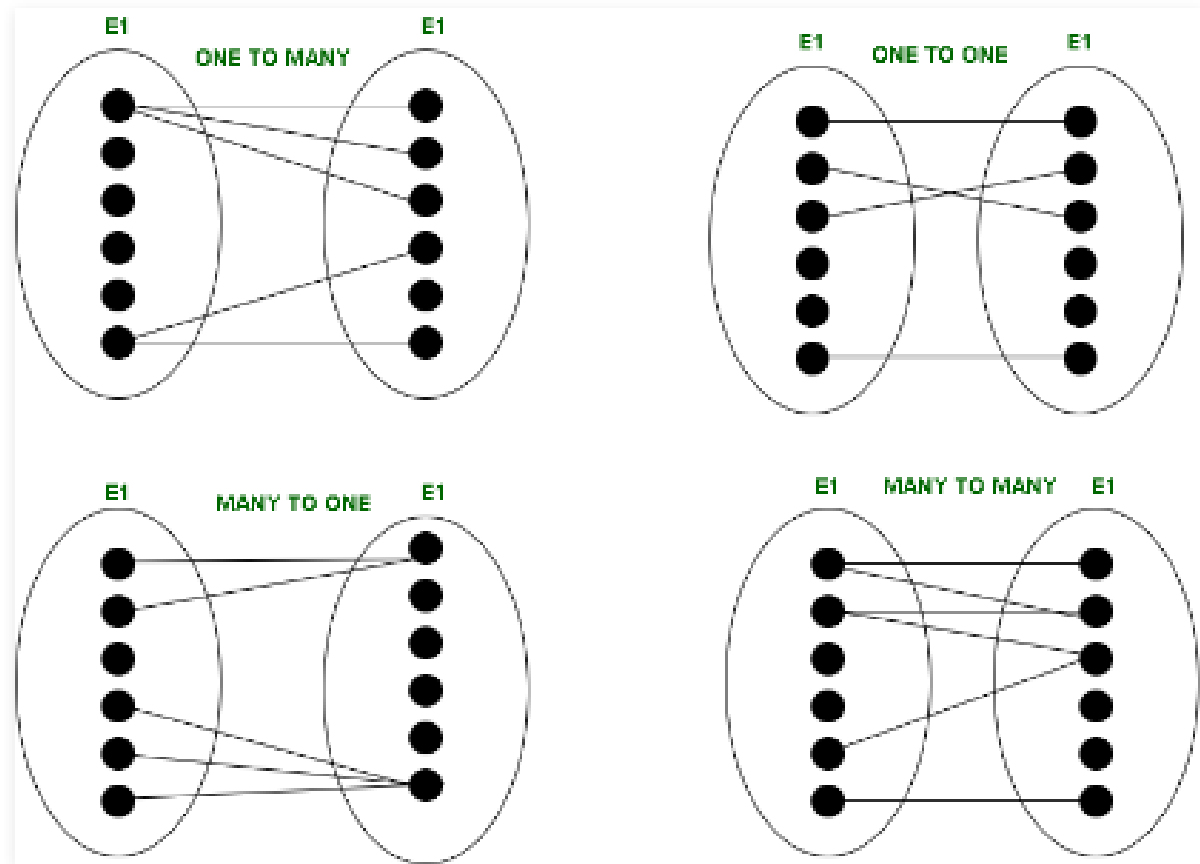
- 1.Cardinality ratio
  - 2.Participation
- 

## Cardinality Ratios for Binary Relationships

Specifies the maximum number of relationship instances that an entity can participate in

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

There are numbers written above the lines which connect relationships and entities. These are called cardinality ratios.



## Cardinality Ratios for Binary Relationships

### 1. **One-to-one (1:1)** –

When one entity in each entity set takes part at most once in the relationship, the cardinality is one-to-one.

### 2. **One-to-many (1: N)** –

If entities in the first entity set take part in the relationship set at most once and entities in the second entity set take part many times (at least twice), the cardinality is said to be one-to-many.

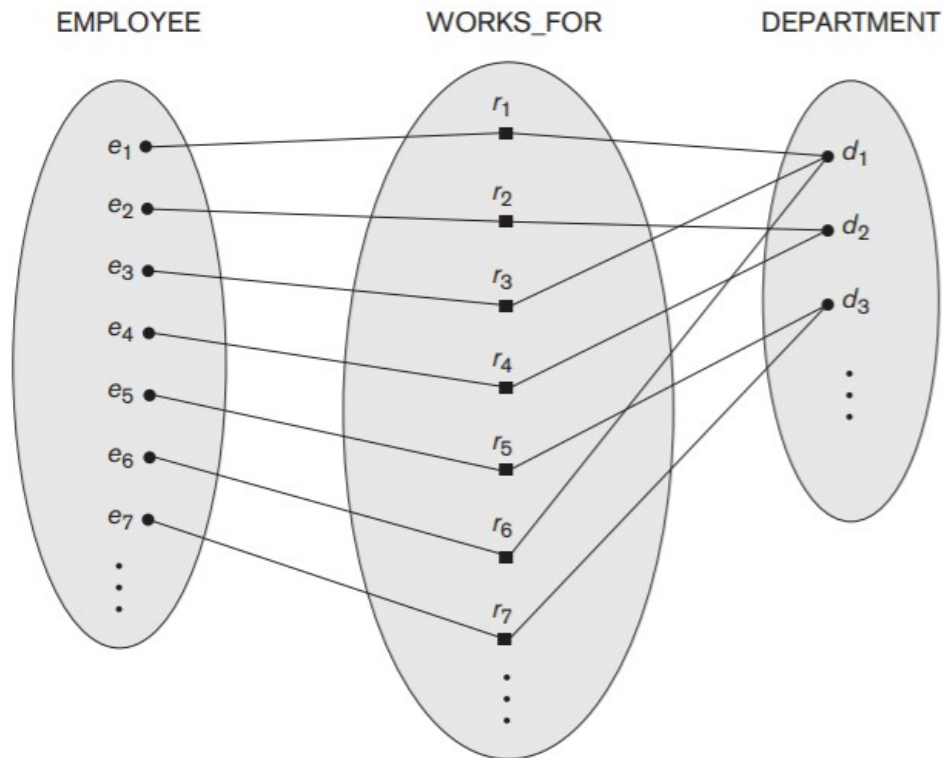
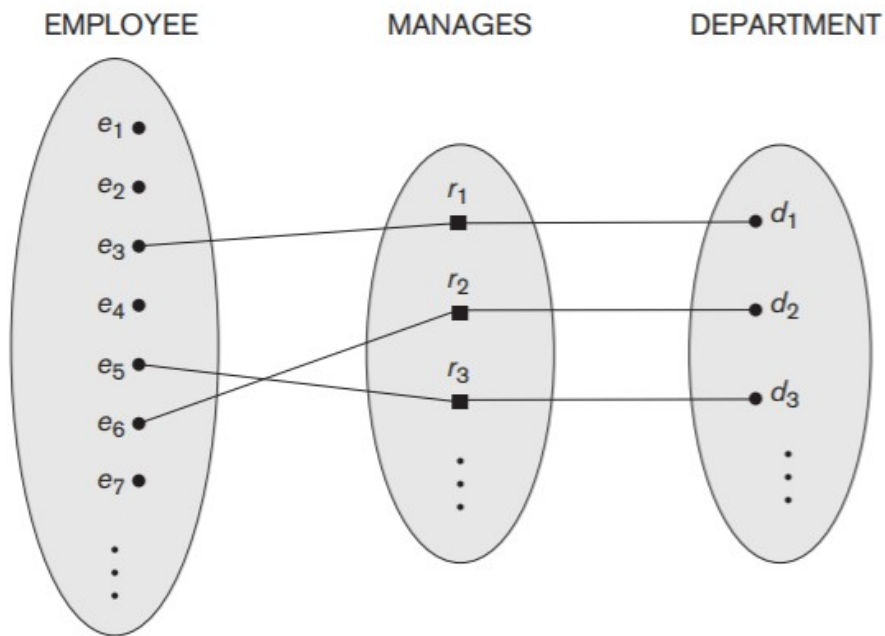
### 3. **Many-to-one (N:1)** –

If entities in the first entity set take part in the relationship set many times (at least twice), while entities in the second entity set take part at most once, the cardinality is said to be many-to-one.

### 4. **Many-to-many (N: N)** –

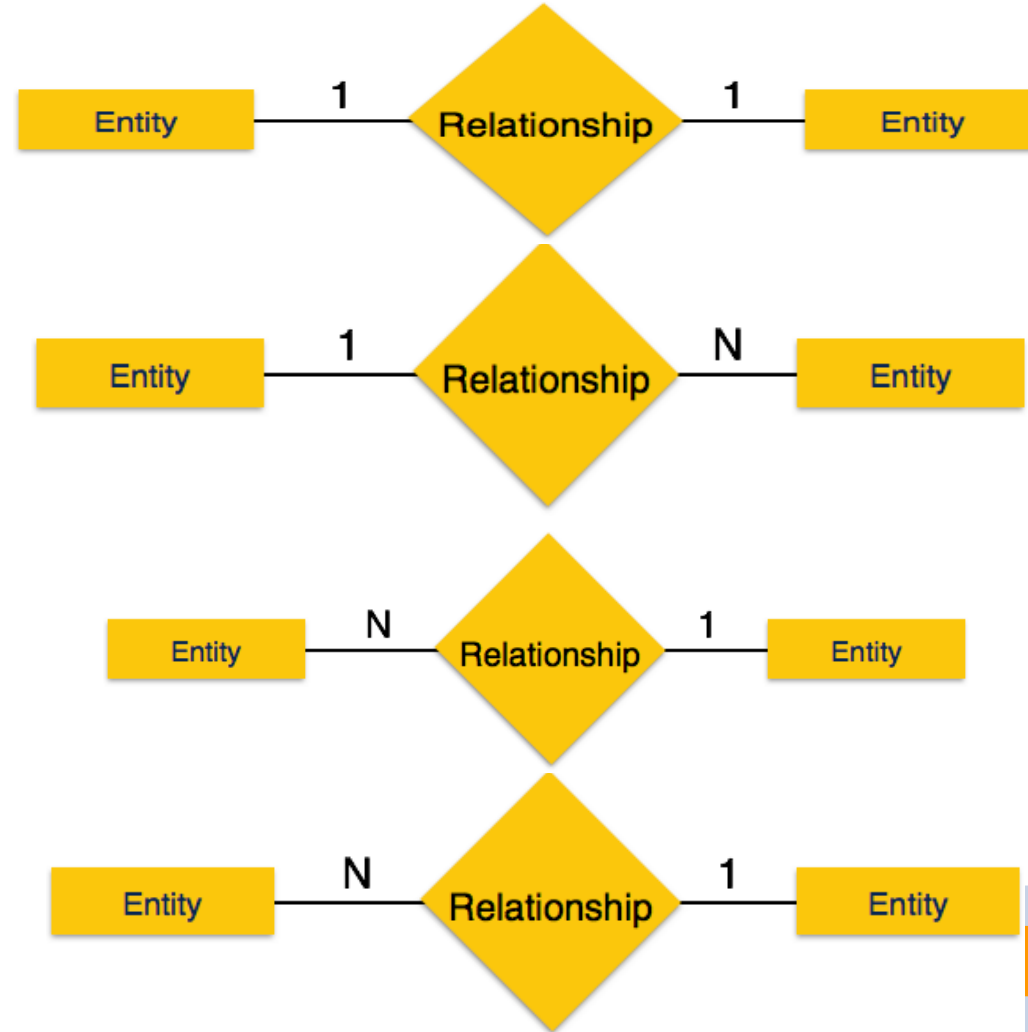
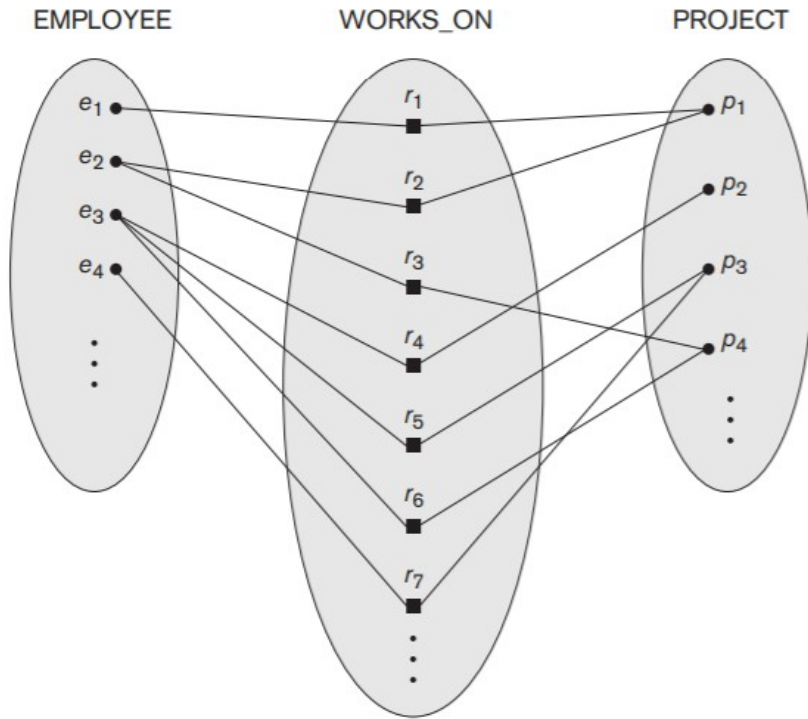
The cardinality is said to be many to many if entities in both the entity sets take part many times (at least twice) in the relationship set.

# Cardinality Ratios for Binary Relationships





# Cardinality Ratios for Binary Relationships

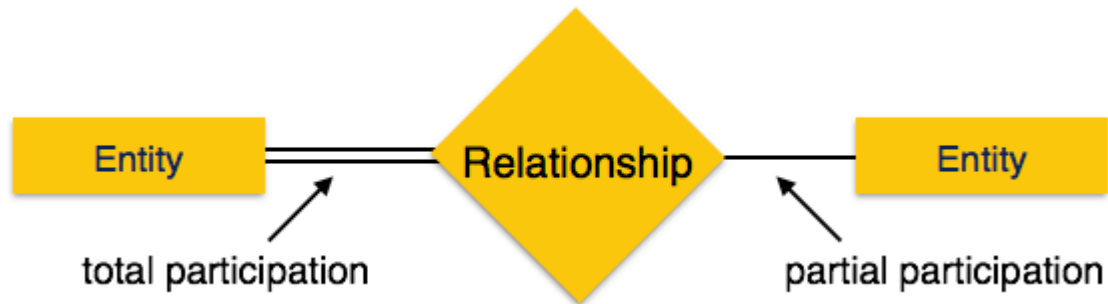


## Participation Constraints

- The participation constraint specifies whether the existence of an entity depends on its being related to another entity via the relationship type
- Specifies the minimum number of relationship instances that each entity can participate in-minimum cardinality constraint

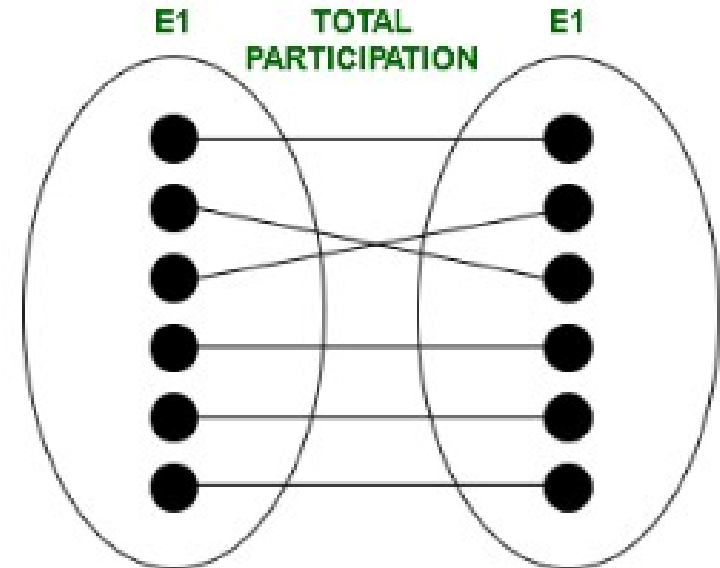
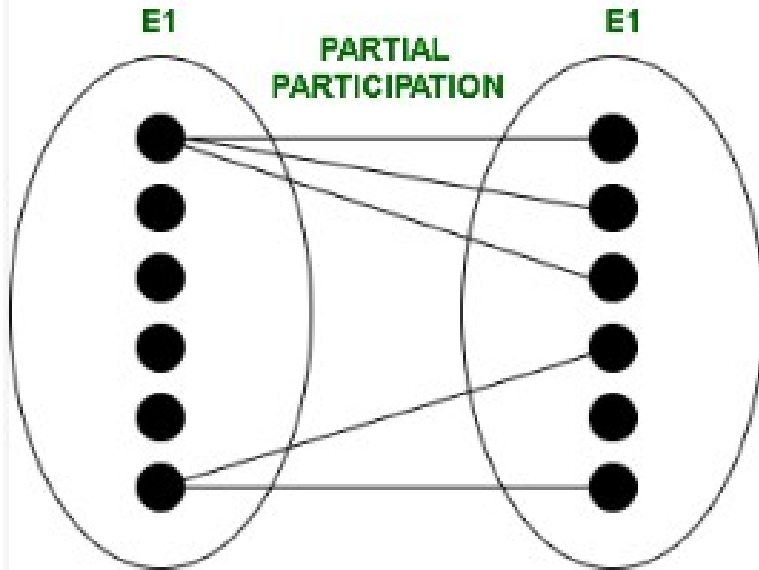
### Types of Participation constraints:

1. Total
2. Partial



## Participation Constraints

When each entity in an entity set participates in a relation, it is called Total Participation. However, when all entities in the given entity set do not participate in a relation, it is called Partial Participation.






## Participation Constraints

### Total participation(existence dependency) constraint:

- Entity can exist only if it participates in at least one relationship instance of a relationship type
- **Example:** If a company policy states that every employee must work for a department, then Participation of **EMPLOYEE in WORKS\_FOR is total participation**

### Partial participation constraint:

- Some or part of the set of entities from one entity type are related to some entities of another entity type via a relationship, but not necessarily all
  - **Example:** Every employee does not manage a department
  - The participation of **EMPLOYEE in the MANAGES relationship type**
- 



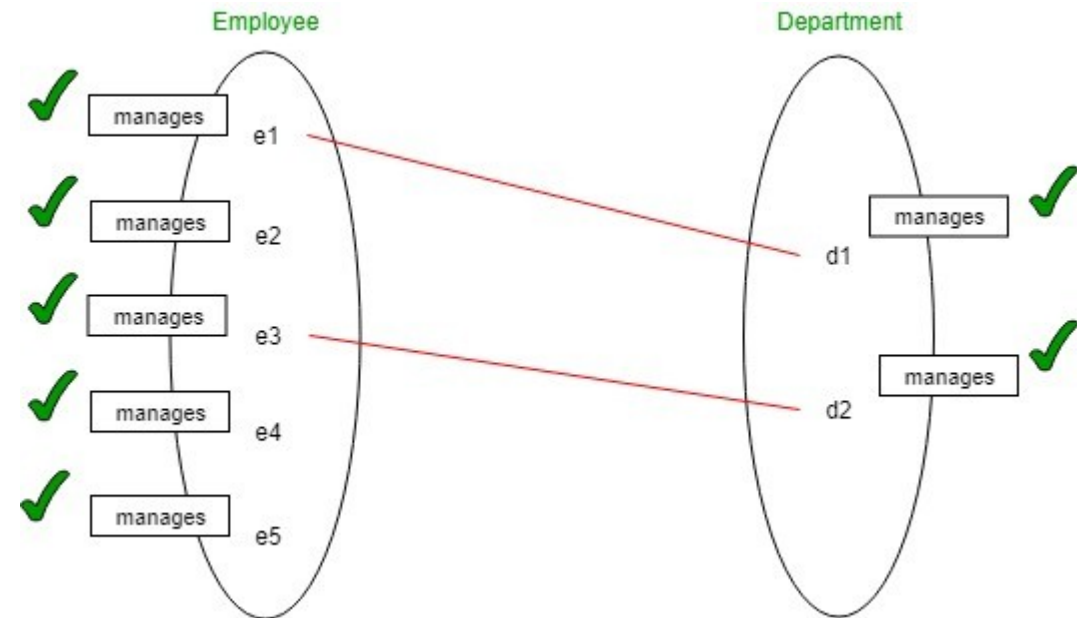
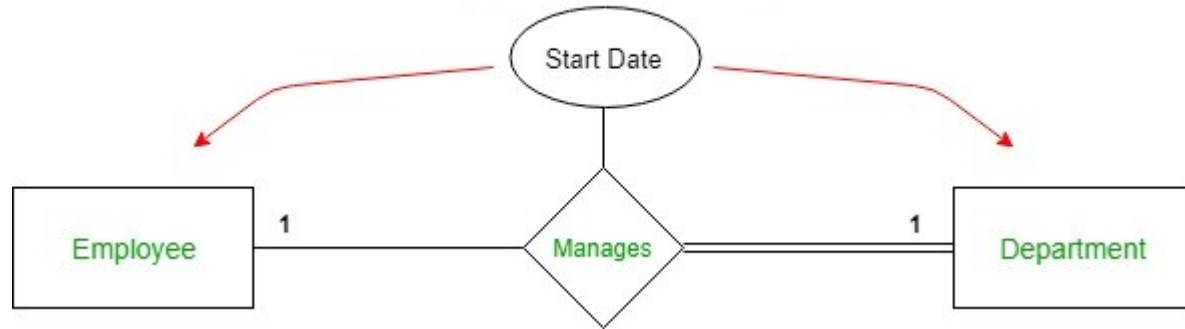
## Attributes to Relationships in ER Model

- In ER model, entities have attributes which can be of various types like single-valued, multi-valued, composite, simple, stored, derived and complex.
- But relationships can also have attributes associated to them.
- Generally it is **not recommended** to give attributes to the relationships if not required because while converting the ER model into Relational model, things may get complex and we may require to create a separate table for representing the relationship.

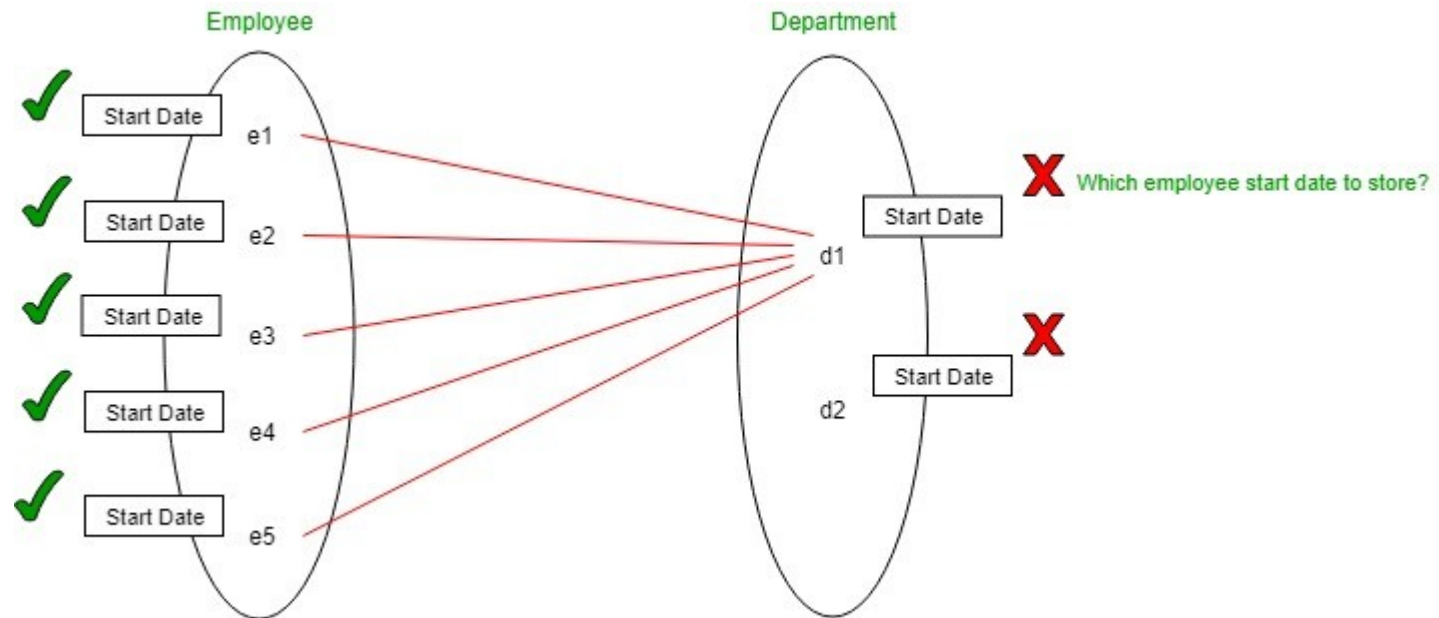
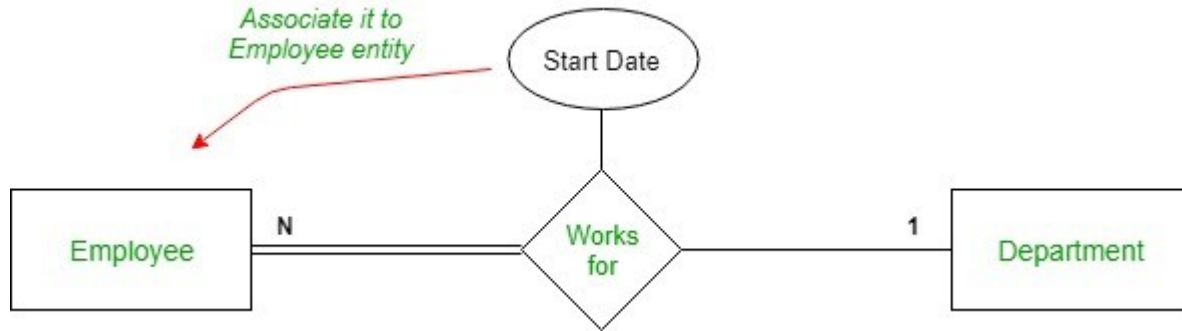


# Attributes to Relationships in ER Model

Associate it to either  
Employee or Department entity

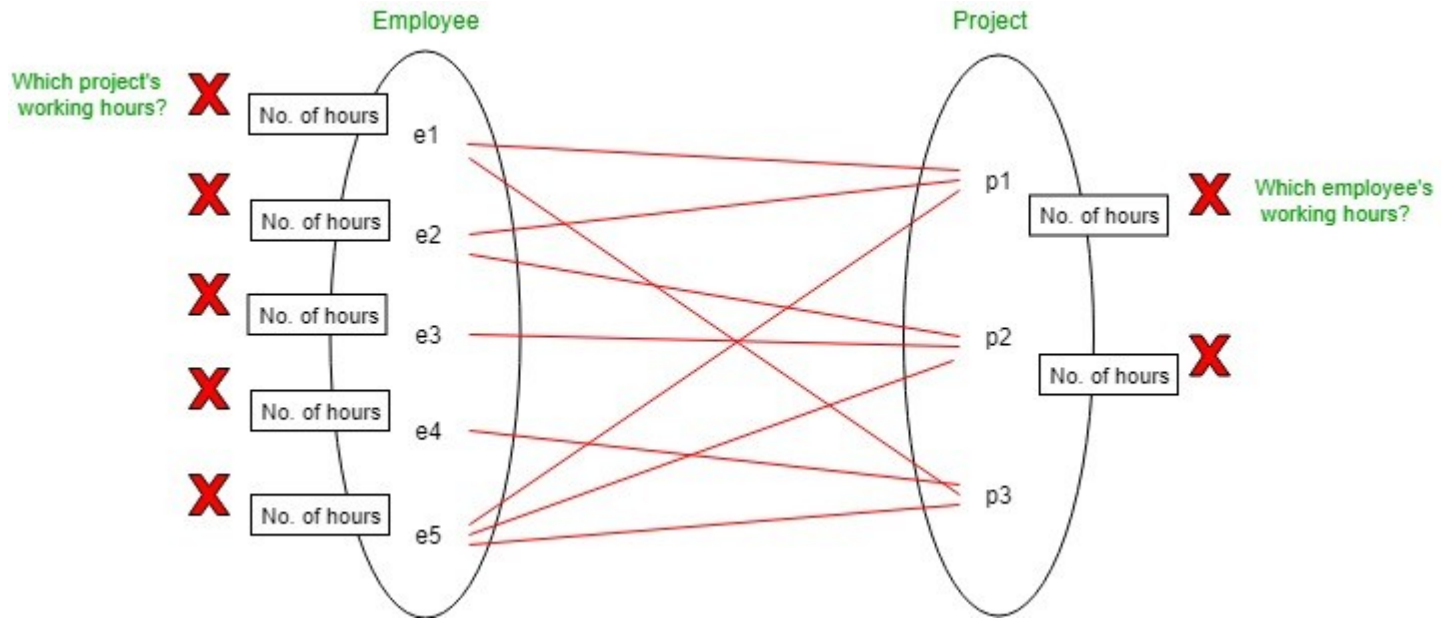
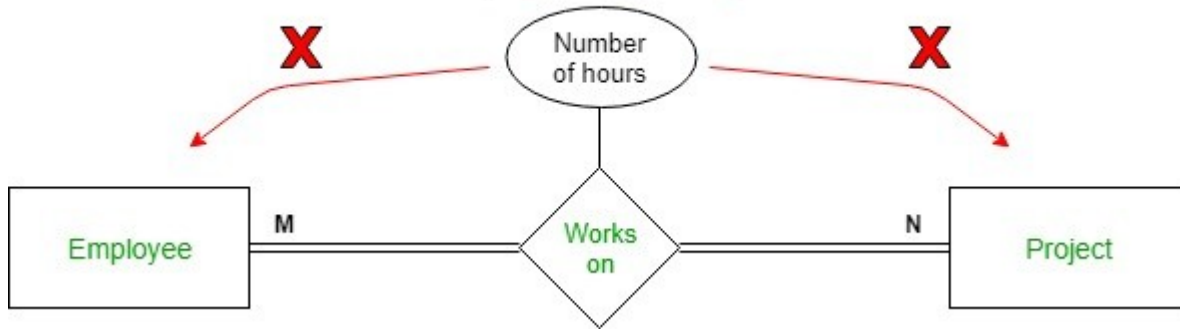


# Attributes to Relationships in ER Model



# Attributes to Relationships in ER Model

Can't associate it to  
Employee or Project entity



Conclusion:

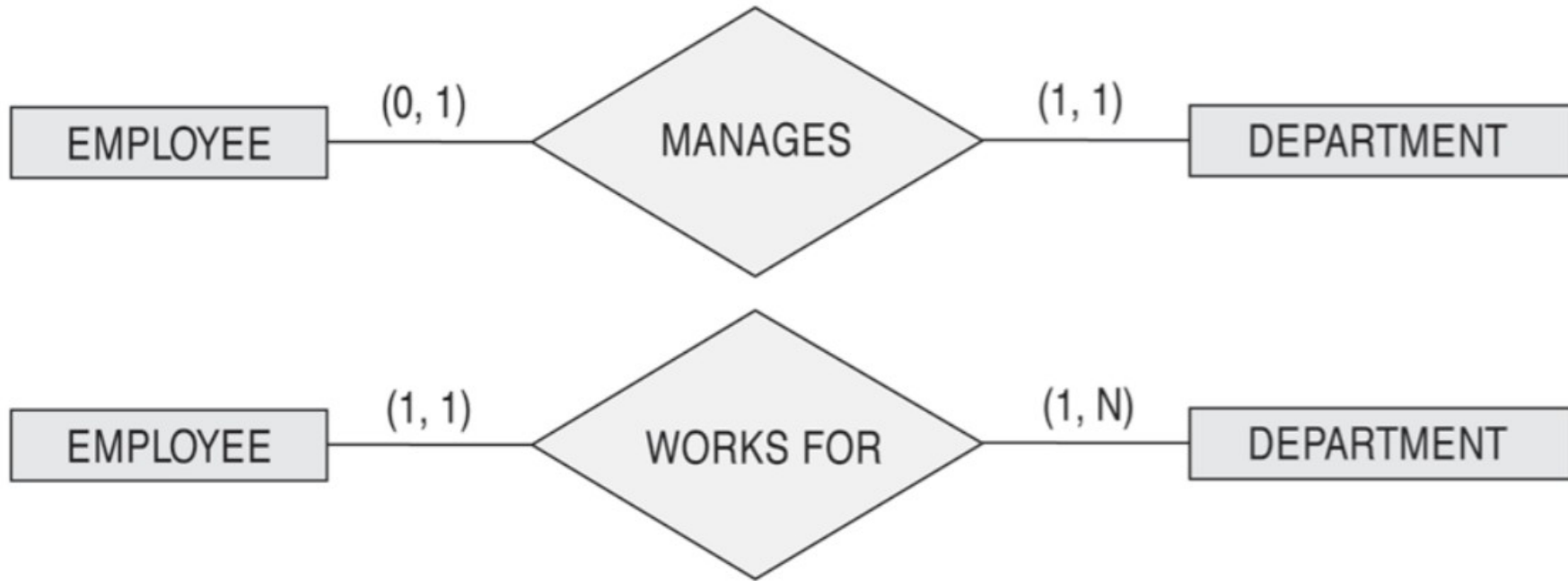
Give attributes to a relationship only in the case of many to many relationship.



## Alternative (min, max) notation for relationship structural constraints:

- Specified on each participation of an entity type E in a relationship type R
- Specifies that each entity e in E participates in at least *min* and at most *max* relationship instances in R
- Default(no constraint): min=0, max=n (signifying no limit)
- Must have  $\text{min} \leq \text{max}$ ,  $\text{min} \geq 0$ ,  $\text{max} \geq 1$
- Derived from the knowledge of mini-world constraints
- Examples:
  - A department has exactly one manager and an employee can manage at most one department.
    - Specify (0,1) for participation of EMPLOYEE in MANAGES
    - Specify (1,1) for participation of DEPARTMENT in MANAGES
  - An employee can work for exactly one department but a department can have any number of employees.
    - Specify (1,1) for participation of EMPLOYEE in WORKS\_FOR
    - Specify (0,n) for participation of DEPARTMENT in WORKS\_FOR

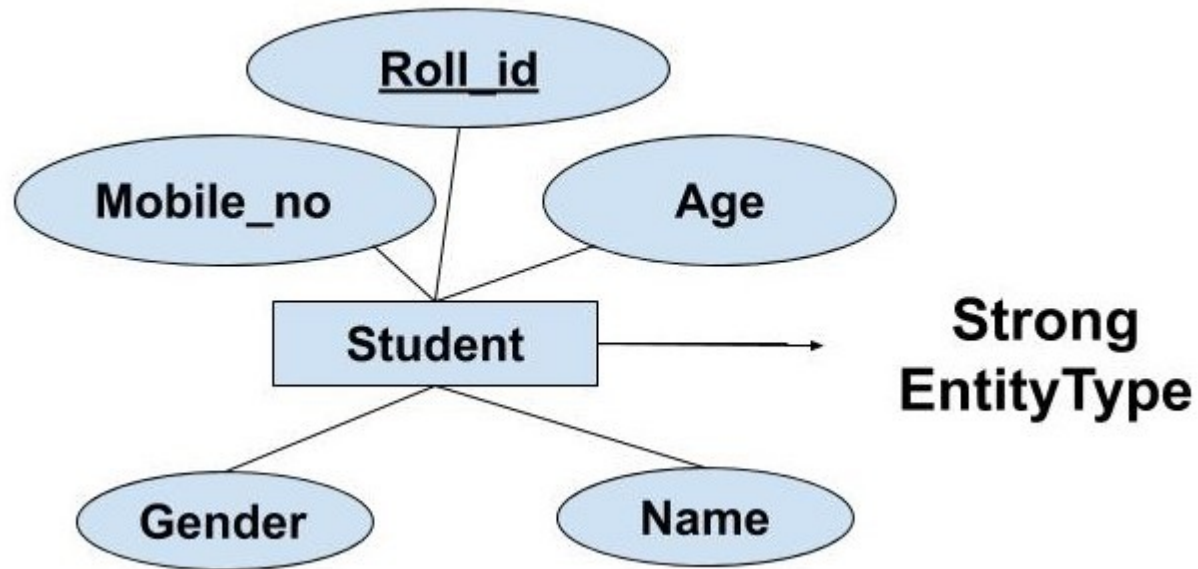
## Alternative (min, max) notation for relationship structural constraints:



# Types of Entity Types

## 1. Strong Entity Type

- Strong entity are those entity types which has a key attribute.
- The primary key helps in identifying each entity uniquely.
- It is represented by a rectangle.






# Types of Entity Types

## 2. Weak Entity Type:

- Weak entity type doesn't have a key attribute.
- Weak entity type can't be identified on its own.
- It depends upon some other strong entity for its distinct identity.


### Examples:

1. There can be children only if the parent exists. There can be no independent existence of children.
  2. There can be a room only if building exists. There can be no independent existence of a room.
  3. A DRIVER\_LICENSE entity cannot exist unless it is related to a PERSON entity, even though it has its own key
- 



# Types of Entity Types

## 2. 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)
  - Not every existence dependency results in a weak entity type
  - The relationship between a weak entity type and strong entity type is called an identifying relationship and shown with a double outlined diamond instead of a single outlined diamond.
- 



# Types of Entity Types

## Strong Entity Set

Strong entity set always has a primary key.

It is represented by a rectangle symbol.

It contains a Primary key represented by the underline symbol.

The member of a strong entity set is called as dominant entity set.


## Weak Entity Set

It does not have enough attributes to build a primary key.

It is represented by a double rectangle symbol.

It contains a Partial Key which is represented by a dashed underline symbol.

The member of a weak entity set called as a subordinate entity set.





# Types of Entity Types

## Strong Entity Set

Primary Key is one of its attributes which helps to identify its member.

In the ER diagram the relationship between two strong entity set shown by using a diamond symbol.


The connecting line of the strong entity set with the relationship is single.

## Weak Entity Set










In a weak entity set, it is a combination of primary key and partial key of the strong entity set.

The relationship between one strong and a weak entity set shown by using the double diamond symbol.

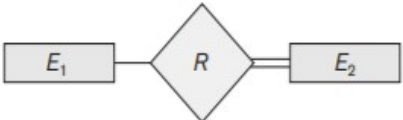
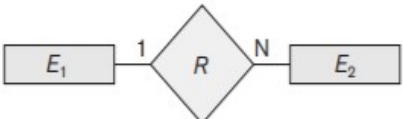
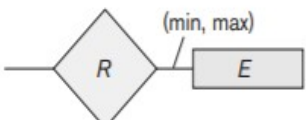
The line connecting the weak entity set for identifying relationship is double.



# Symbols used in ER Diagrams

Symbol	Meaning
	Entity
	Weak Entity
	Relationship
	Identifying Relationship
	Attribute
	Key Attribute
	Multivalued Attribute
	Composite Attribute
	Derived Attribute

	Total Participation of $E_2$ in $R$
	Cardinality Ratio 1 : N for $E_1 : E_2$ in $R$
	Structural Constraint (min, max) on Participation of $E$ in $R$



# How to Create an Entity Relationship Diagram (ERD)



## Steps to Create an ER Diagram

### Problem Statement (Example)

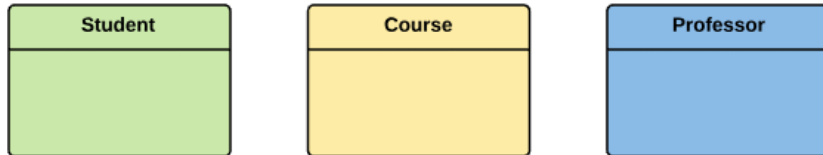
In a university, a Student enrolls in Courses. A student must be assigned to at least one or more Courses. Each course is taught by a single Professor. To maintain instruction quality, a Professor can deliver only one course

# How to Create an Entity Relationship Diagram (ERD)

## Step 1) Entity Identification

We have three entities

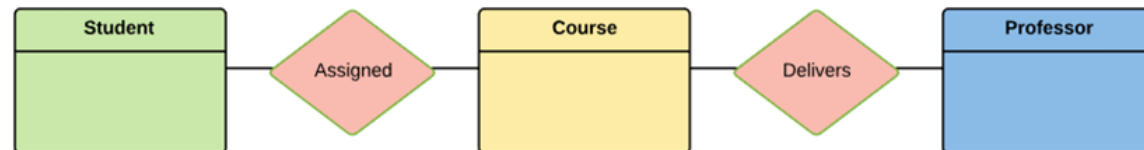
- Student
- Course
- Professor



## Step 2) Relationship Identification

We have the following two relationships

- The student is assigned a course
- Professor delivers a course



# How to Create an Entity Relationship Diagram (ERD)

## Step 3) Cardinality Identification

For them problem statement we know that,

- A student can be assigned **multiple** courses
- A Professor can deliver only **one** course

## Step 4) Identify Attributes

Entity	Primary Key	Attribute
Student	Student_ID	StudentName
Professor	Employee_ID	ProfessorName
Course	Course_ID	CourseName

## Step 5) Create the ERD Diagram



# How to Create an Entity Relationship Diagram (ERD)

## Problem

Drawing of ER model of university database application considering the constraints –

A university has many departments.

Each department has multiple instructors (one person is HOD). Here the HOD refers to the head of department.

An instructor belongs to only one department.

Each department offers multiple courses, each subject is taught by a single instructor.

A student may enroll for many courses offered by different departments.





# How to Create an Entity Relationship Diagram (ERD)

## Solution

Step 1 – Identifying the entity sets.

Step 2 – Identifying the attributes for the given entities

Step 3 – Identifying the Key attributes

Step 4 – Identifying the relationship between entity sets

Step 5 – Complete ER model





# How to Create an Entity Relationship Diagram (ERD)

## Step 1 – Identifying the entity sets.

The entity set has multiple instances in a given business scenario.

As per the given constraints the entity sets are as follows –

- Department
- Course
- Student
- Instructor

Head of the Department (HOD) is not an entity set. It is a relationship between the instructor and department entities.





# How to Create an Entity Relationship Diagram (ERD)

## Step 2 – Identifying the attributes for the given entities

**Department** (department Name and location ).

**Course** (courseNo, course Name, Duration, and prerequisite).

**Instructor** (Instructor Name, Room No, and telephone number).

**Student** (Student No, Student Name, and date of birth).

## Step 3 – Identifying the Key attributes

**Department** (department Name and location ).

**Course** (courseNo, course Name, Duration, and prerequisite).

**Instructor** (Instructor Name, Room No, and telephone number).

**Student** (Student No, Student Name, and date of birth).



# How to Create an Entity Relationship Diagram (ERD)

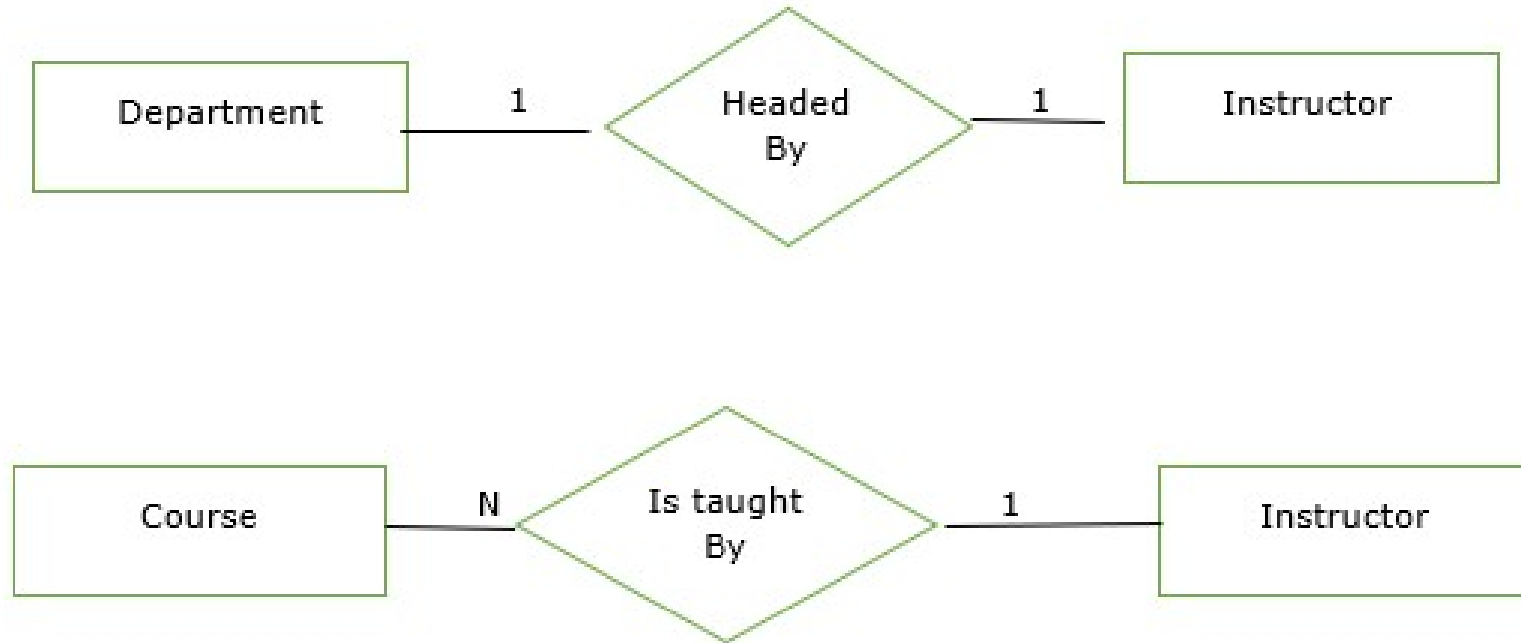
## Step 4 – Identifying the relationship between entity sets





# How to Create an Entity Relationship Diagram (ERD)

## Step 4 – Identifying the relationship between entity sets






# How to Create an Entity Relationship Diagram (ERD)

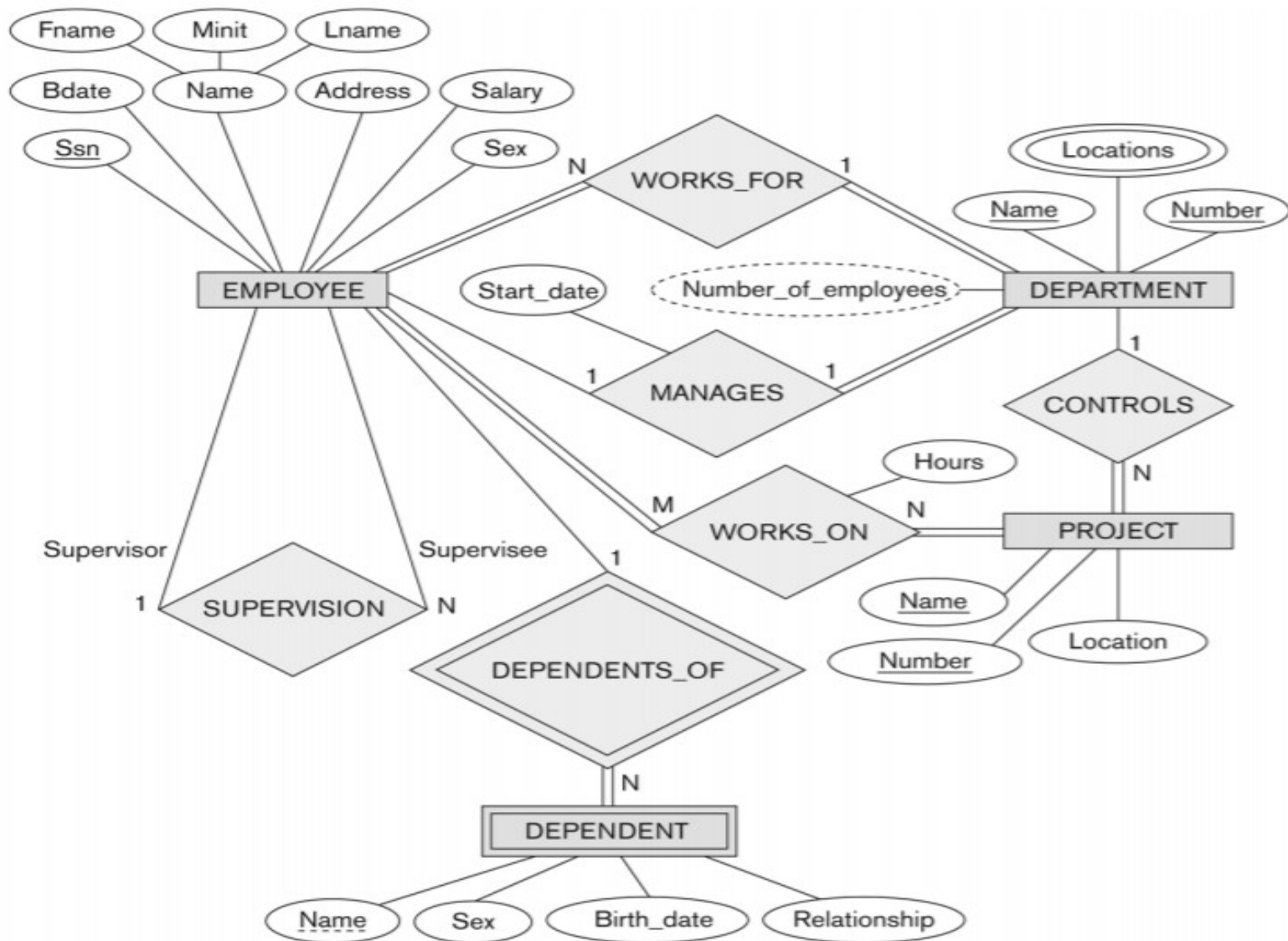
**Step 5 – Complete ER model**





# Entity Relationship Diagram Examples

- Company organized into DEPARTMENT. Each department has unique name and a particular employee who manages the department. Start date for the manager is recorded. Department may have several locations.
  - A department controls a number of PROJECT. Projects have a unique name, number and a single location.
  - Company's EMPLOYEE name, ssno, address, salary, sex and birth date are recorded. An employee is assigned to one department, but may work for several Projects (not necessarily controlled by her dept).
  - Number of hours/week an employee works on each project is recorded; The immediate supervisor for the employee.
  - Employee's DEPENDENT are tracked for health insurance purposes (dependent name, birthdate, relationship to employee).
- 





# Entity Relationship Diagram Examples

1. Suppose you are given the following requirements for a simple database for the **National Hockey League (NHL)**:

- The NHL has many teams,
- Each team has a name, a city, a coach, a captain, and a set of players,
- Each player belongs to only one team,
- Each player has a name, a position (such as left wing or goalie), a skill level, and
- A set of injury records,
- A team captain is also a player,
- A game is played between two teams (referred to as `host_team` and `guest_team`) and has a date (such as May 11th, 1999) and a score (such as 4 to 2).

Construct a clean and concise ER diagram for the NHL database





## Entity Relationship Diagram Examples

2. Construct an E-R diagram for a car-insurance company whose customers own one or more cars each. Each car has associated with it zero to any number of recorded accidents.
3. Construct an E-R diagram for a hospital with a set of patients and a set of medical doctors. Associate with each patient a log of the various tests and examinations conducted.
4. Consider a database used to record the marks that students get in different exams of different course offerings. Construct an E-R diagram that models exams as entities, and uses a ternary relationship, for the above database.

