database

E-R model

2024-2025

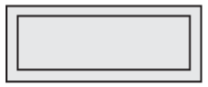
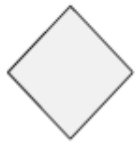
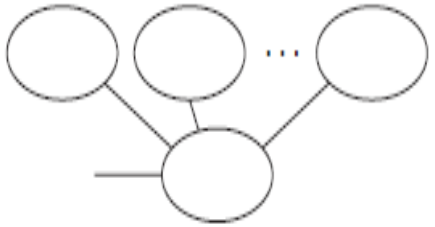
8.The Entity Relationship Mode

ER data model has existed for over 35 years. It is well suited to data modelling for use with DB because it is fairly abstract and is easy to discuss and explain. ER models are readily translated to relations. ER models, also called an ER schema, are represented by

ER diagrams. ER modelling is based on two concepts:

1. Entities, defined as tables that hold specific information (data)
2. Relationships, defined as the associations or interactions between entities

Table(1) Entity Relationship Diagram Symbols

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

- **Entity**
- An entity is an object in the real world with an independent existence that can be differentiated from other objects. An entity might be
  - An object with physical existence (e.g., a lecturer, a student, a car)
  - An object with conceptual existence (e.g., a course, a job, a position)

- Entities can be classified based on their strength into:-

-**Weak entity:** an entity is considered **weak** if its tables are existence dependent.
- That is, it cannot exist without a relationship with another entity
- Its primary key is derived from the primary key of the parent entity

- Strong entity: an entity is considered **strong** if it can exist a part from all of its related entities.

- A table without a foreign key or a table that contains a foreign key that can contain nulls is a strong entity

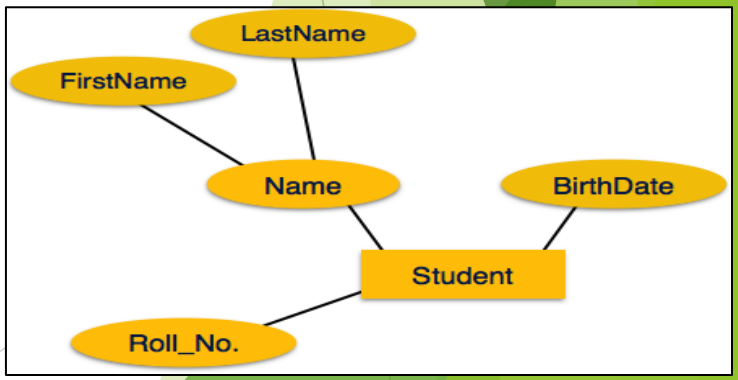
Entity and attributes

- 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.
- **Types Of Attributes**

Several types of attributes occur in the ER model: **simple** versus **composite**, **single** valued **versus multivalued**, and **stored** versus **derived.**

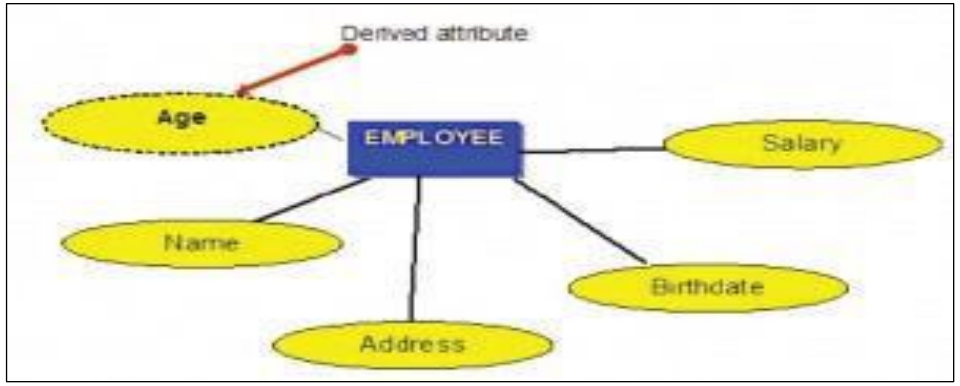
1. Simple attributes: Are those drawn from the atomic value domains; they are also called single-valued attributes. In the COMPANY database, an example of this would be: Name = {John} ; Age = {23}

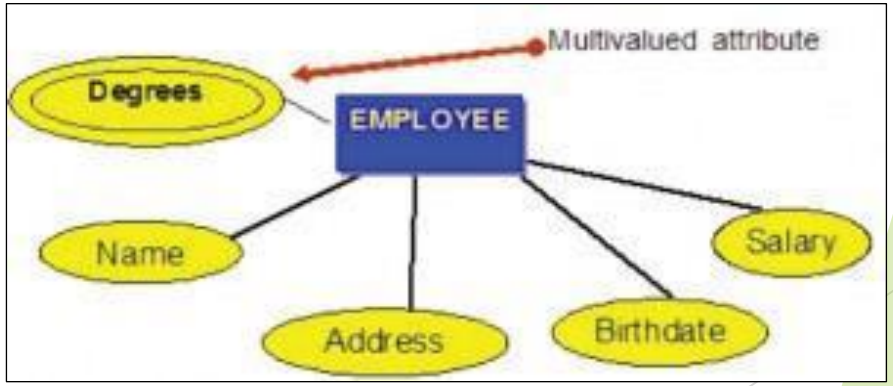
2. Composite attributes: Are those that consist of a hierarchy of attributes. Figure 3 Address may consist of Number, Street and Suburb. So this would be written as → Address = {59 + 'Meek Street' + 'Kingsford'}



3. Multivalued attributes: Are attributes that have a set of values for each entity. For ex: the degrees of an employee: BSc, MIT, PhD.

4. Derived attributes: Are attributes that contain values calculated from other attributes. Age can be derived from the attribute Birthdate. In this situation, Birthdate is called a stored attribute, which is physically saved to the database.



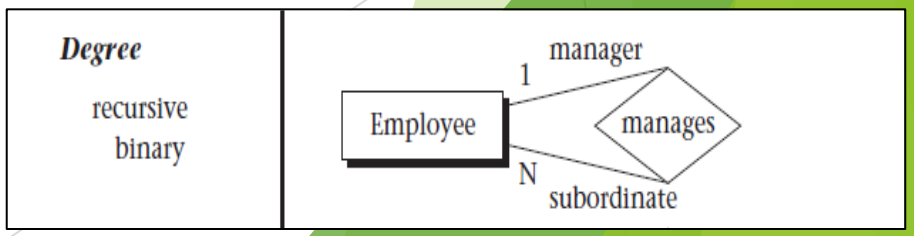


Degree of a Relationship

- The degree of a relationship is the number of entities associated in the relationship. **Binary** and **ternary** relationships are special cases where the degrees are 2 and 3, respectively.

- The **binary** relationship, an association between two entities, is by far the most common type in the natural world. In fact, many modeling systems use only this type. we see many examples of the association of two entities in different ways: Department and Division, Department and Employee, Employee and Project, and so on.

- A binary recursive relationship (for example, “manages”) relates a particular Employee to another Employee by management. It is called **recursive** is one in which the same entity participates more than once in the relationship; because the entity relates only to another instance of its own type.
- The binary recursive relationship construct is a diamond with both connections to the same entity. A ternary relationship is an association among three entities.

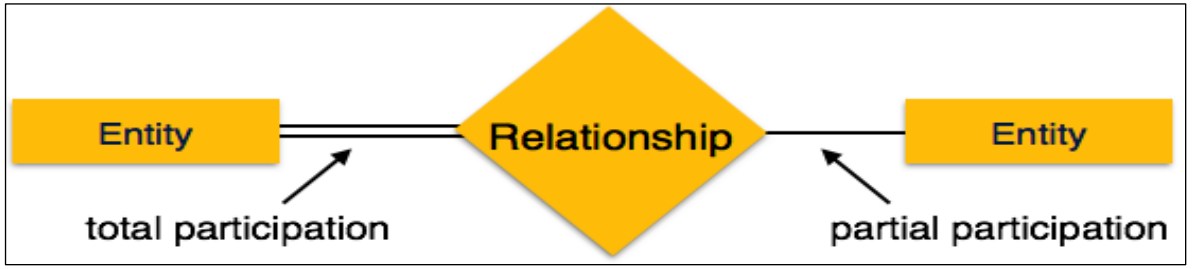


Participation

- Participation refers to whether an entity *must* participate in a relationship with another entity to exist. can be total or partial (optional):

- **Total participation** is where an entity must participate in a relationship to exist. For example, an employee must work for at least one department to exist as an employee.

- **Partial (optional) participation** is where the entity can exist without participating in a relationship with another entity . For example, the entity course may exist within an organization, even though it has no current students.



9.How to Convert ER Diagram to Relational Database

- We will be following the simple rules:

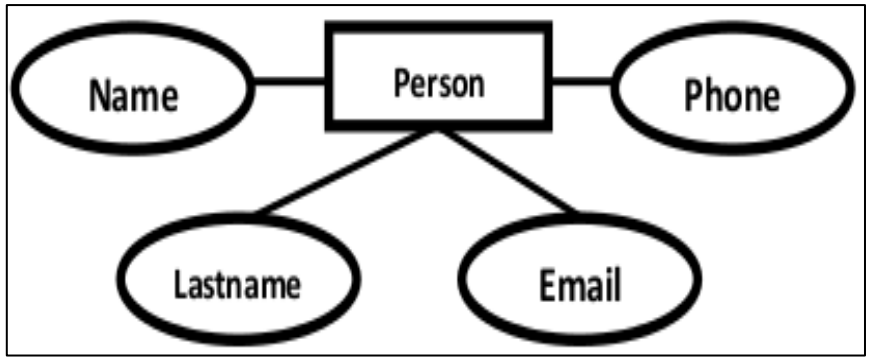
1. **Entities and Simple Attributes:**

- An entity type within ER diagram is turned into a table. You may preferably keep the same name for the entity or give it a sensible name but avoid DBMS reserved words as well as avoid the use of special characters.
- Each attribute turns into a column (attribute) in the table. The key attribute of the entity is the primary key of the table which is usually underlined. It can be composite if required but can never be null.
- It is highly recommended that every table **should start with its primary key** attribute conventionally named as TablenameID.

- Taking the following simple ER diagram:

The initial relational schema is expressed in the following format writing the table names with the attributes list inside a parentheses as shown below for Persons and Phones are Tables. name, lastname, are Table Columns (Attributes).

- Persons (**personid**, name, lastname, email)



2. Multi-Valued Attributes

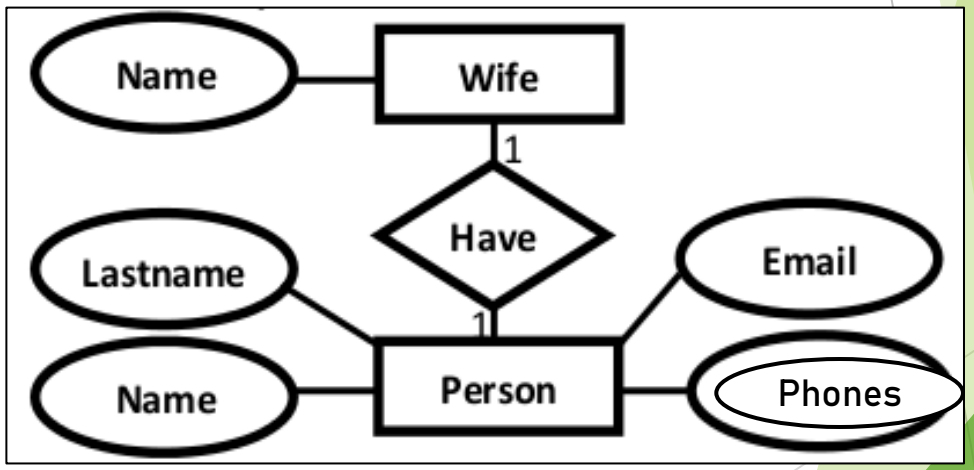
A multi-valued attribute is usually represented with a double-line oval.

If you have a multi-valued attribute, take the attribute and turn it into a new entity or table of its own. Then make a 1: N relationship between the new entity and the existing one. In simple words:

- Create a table for the attribute.
- Add the primary (id) column of the parent entity as a foreign key within the new table

Persons (personid, name, lastname, email)

Phones (phoneid , ***personid***, phone)



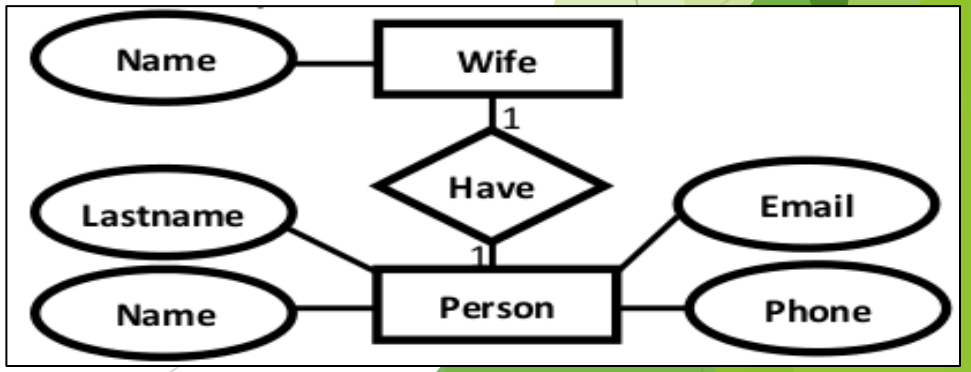
3. 1:1 Relationships

- To keep it simple and even for better performances at data retrieval, I would personally recommend using attributes to represent such relationship. For instance, let us consider the case where the Person has or optionally has one wife. You can place the primary key of the wife within the table of the Persons which we call in this case Foreign key as shown below.

Persons (personid, name, lastname, email , phone, ***wifeid***)
Wife (wifeid , name)

- Or vice versa to put the personid as a foreign key within the Wife table as shown below:

Persons (personid , name, lastname, email, phone)
Wife (wifeid , name , ***personid)***



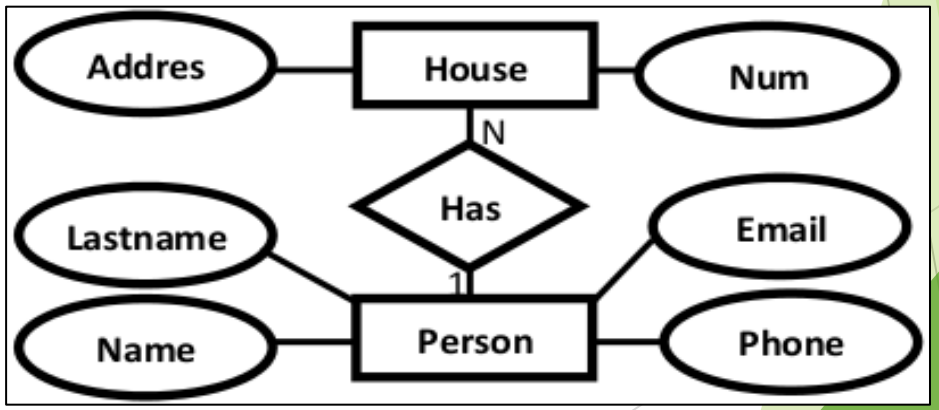
4. 1: N Relationships

- This is the tricky part! For simplicity, use attributes in the same way as 1:1 relationship but we have only one choice as opposed to two choices. For instance, the Person can have a **House** from zero to many, but a **House** can have only one **Person**. To represent such relationship the **personid**as the Parent node must be placed within the Child table as a foreign key but not the other way around as shown next:

- It should convert to:

Persons (personid , name, lastname, email)

House (houseid , num , address, ***personid***)



5. N:N Relationships

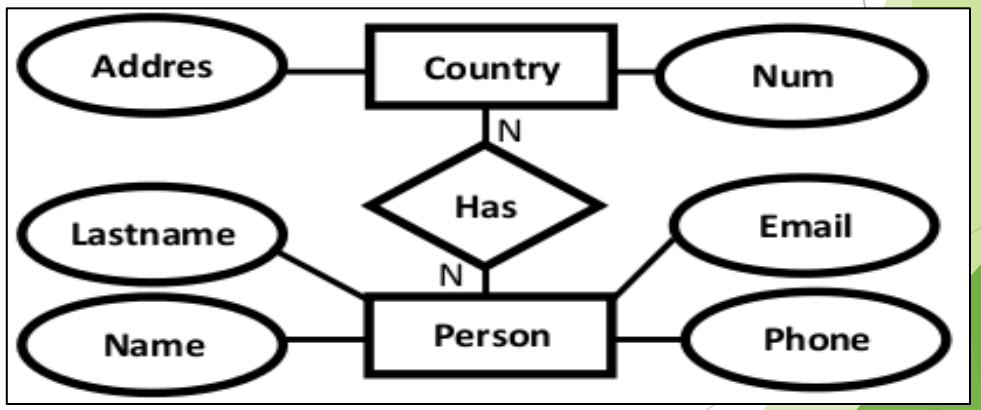
- We normally use tables to express such type of relationship. It is the same for N − ary relationship of ER diagrams. For instance, The Person can live or work in many countries. Also, a country can have many people. To express this relationship within a relational schema we use a separate table as shown below:

- it should convert into :

Persons(personid , name, lastname, email)

Countries (countryid , name, code)

HasRelat (hasrelatid , **personid , countryid**)



Case Study

- We will be producing the relational schema for the following ER diagram: