Template:

ER Diagrams

→ Entity Set

→ Entity : real world object

→ Entity set : Collection of similar objects

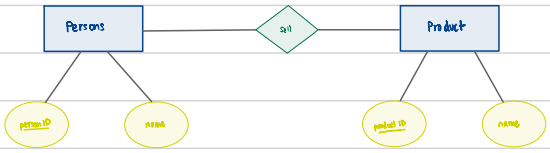
→ Attribute

→ Property of an entity set

→ Primary key : have underline →

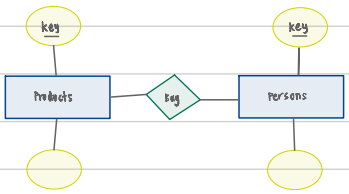
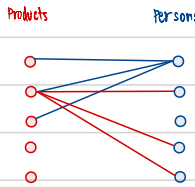
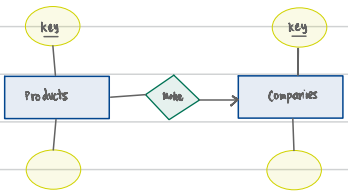
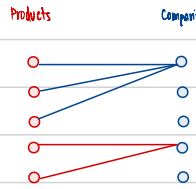
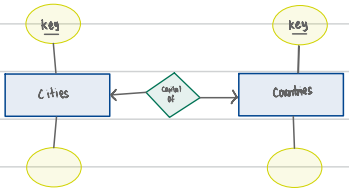
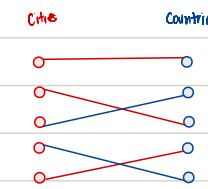
→ Relationship → Connection between 2 entity sets

Combined Example:

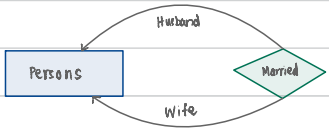


- Person sells product
- Product sold by Persons

Types of Relationships

Many to Many			→ One person can buy multiple products → One product can be bought by multiple persons
Many to One			→ One company can make multiple products → One product can only be made by 1 company → "→" to be placed on "one" side
One to One			→ A city can be the capital of only one country → A country can have only one capital city

Roles

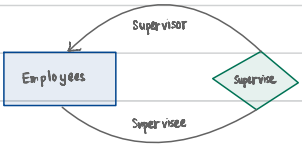


"some persons are married to each other"

Husband	Wife
Bob	Alice
⋮	⋮

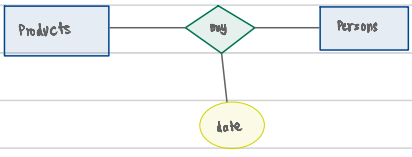
→ sometimes entity set appear > 1

→ Role of person is specified on the edge connecting the entity sets



"some employee supervises other employees"

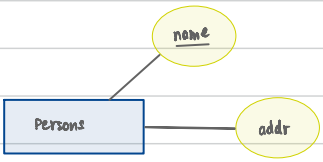
→ without roles, it is unclear whether the many-to-one from supervisees to supervisors, or the other way around



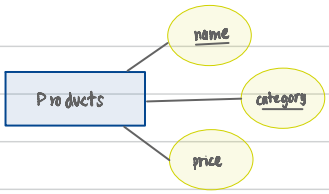
"record the date of purchase"

→ A relationship can have its own attribute

Keys



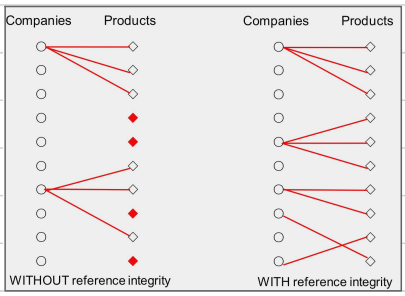
→ A key can have one or more attributes
→ Indicated with an underline
→ UNIQUELY REPRESENT each entity in the entity set
→ Rule: Every entity set should have a key
→ E.g. There can be products with the same name / category, but not both



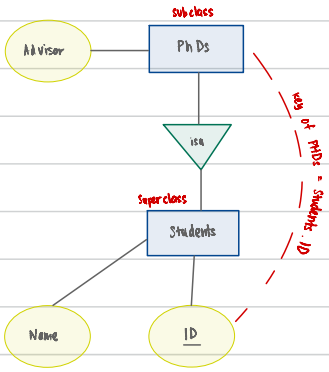
Referential Integrity

→ In this example, every product must be involved in the Make relationship
→ represented by rounded arrow →

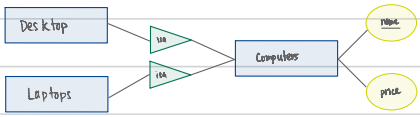
→ Referential Integrity constraint can only apply to the "one" side of many-to-one or one-to-one relationships (cannot be applied to many-to-many relationship)
↳ makes use of "degree constraint" instead



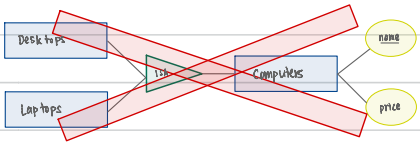
Subclasses



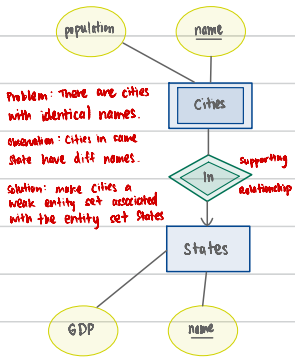
→ Subtype = Special type
→ Connection between subclass and superclass captured by ISA relationship, represented by a triangle
→ key of subclass = key of superclass
→ Entity set can have multiple subclasses



DO NOT DRAW IT THIS WAY



Weak Entity Sets



→ weak entity sets are a special type of entity sets that → cannot be uniquely identified by their own attributes
→ needs attributes from other entities to identify themselves
→ Weak entity set = Double-lined rectangle
→ Supporting relationship = Double-lined diamond
→ The key of Cities = (State.name, Cities.name)

Subclasses VS Weak Entity Sets

→ Subclasses inherits all attributes from superclass
→ Weak Entity Sets just need attributes from supporting entity set to help identify its different entity instances.

Based on example diagrams above: PhDs are a special type of Students.
Cities are NOT a special type of States.

ER diagram design principles

1. Identify the objects involved in application
2. Model each type of objects as an entity set
3. Identify attributes (adjectives) of each entity set
4. Identify the relationships (verbs) among entity sets
5. Refine your design.

Design Principle 1 : Be faithful

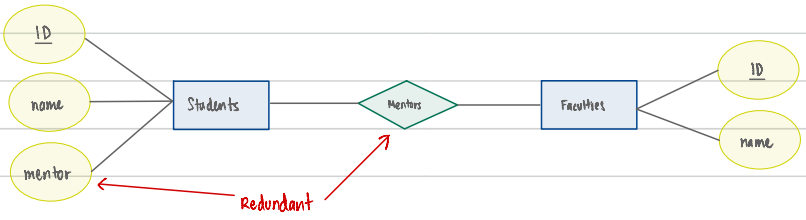
→ Be faithful to the specifications of the application ; capture requirements as much as possible

Design Principle 2 : Avoid Redundancy

→ Avoid repetition of information

→ Problems that may arise :

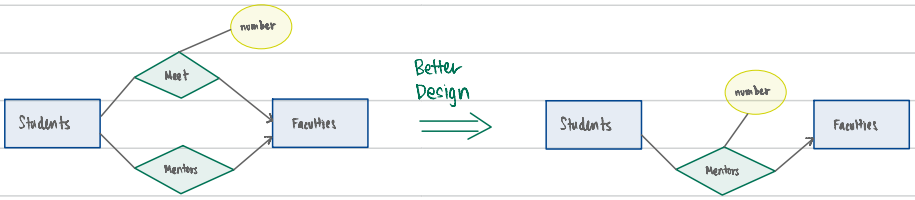
- Waste of space
- Possible inconsistency → gets updated in one place, other does not



Design Principle 3 : Keep It Simple

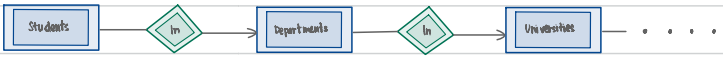
→ Example :

Each student is mentored by one faculty.
One faculty can mentor multiple students.
Record no. of times a mentee meets with mentor



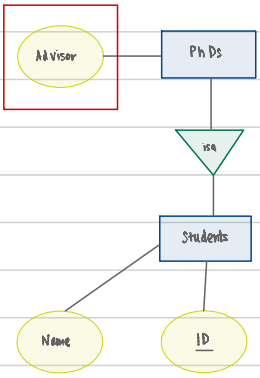
Design Principle 4 : Don't over-use weak entity sets

→ Too many entity sets that should not be "weak"

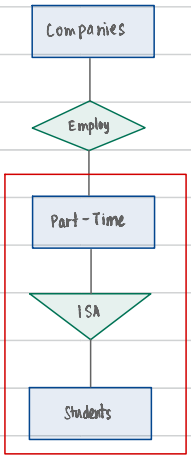


Tips : When to use subclasses

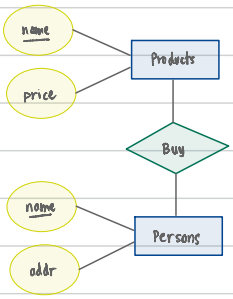
Case 1 : When a subclass has some attribute that is absent from the superclass



Case 2 : When a subclass has its own relationship with some other entity sets



ER diagram → Relational schema



Products (name , price)
Persons (name , addr)
Buy (product_name , person_name)

⇨ Relational schema:
name of table + attribute names

Database schema:
a set of relation schemas

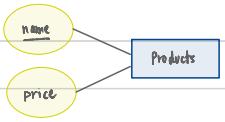
General rules:
- → Each entity set becomes a relation
- → Each many-to-many relationship becomes a relation

Special treatment needed for:
- → Weak Enitity Sets
- → Subclasses
- → Many-to-one & one-to-one relationships

Entity Set → Relation

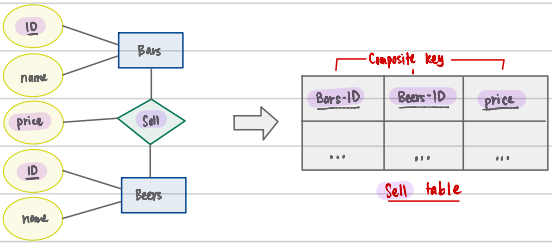
→ Each entity set is converted into a relation that contains all its attributes

→ key of the relation = key of entity set



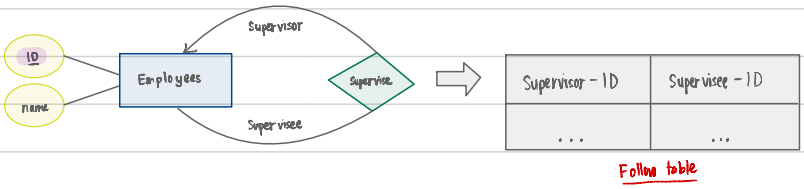
⇨ Products (name , price)

Many-to-Many Relationship → Relation



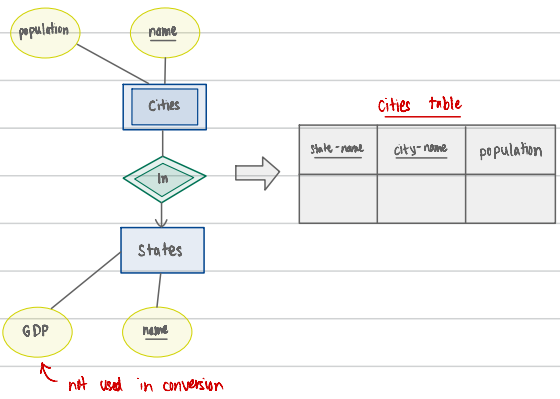
→ Converted into a relation that contains
- all keys of the participating entity sets
- the attributes of the relationship

→ key of relation = keys of the participating entity sets



→ If an entity is involved multiple times in a relationship
- Its key will appear in the corresponding relation multiple times
- The key is re-named according to the corresponding role

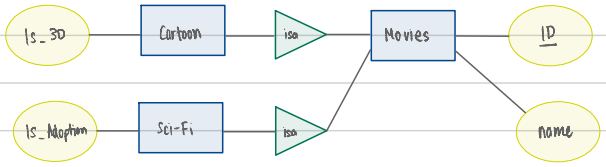
Weak Entity Set → Relation



→ Each weak entity set is converted to a relation that contains
- all of its attributes, and
- key attributes of the supporting entity set

→ Supporting relationship ignored

Subclass → Relation



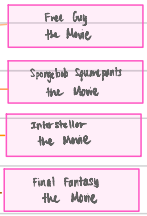
→ 3 approaches:

- ER approach — one record, multiple relations
- OO approach — one record, one relation (potentially many multiple relations)
- NULL approach — one big relation, lots of NULLs

The OO approach

→ One relation for each entity set & each possible subclass combination

- Movies (ID, name)
- Cartoon (ID, Is_3D)
- Sci-Fi (ID, Is_Adoption)
- Sci-Fi – Cartoon (ID, name, Is_3D, Is_Adoption)



→ All subclass become schema, inheriting everything from superclass
↳ therefore, don't need to be placed in superclass table

Example

→ Advantage: Good for searching subclass combination
→ Disadvantage: May have too many tables

The NULL approach

→ One relation that includes everything

- Movies (ID, name, Is_3D, Is_Adoption)

→ for non-cartoon movies, "Is_3D" = NULL
→ For non-sci-fi movies, "Is_Adoption" = NULL

→ Advantage: needs only one relation
→ Disadvantage: May have many NULL values

The ER Approach

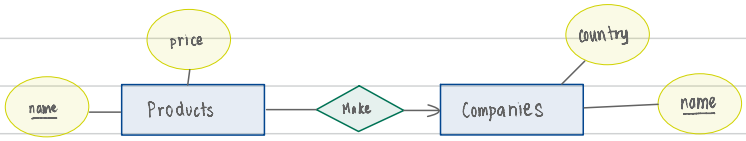
→ one relation for each entity set

- Movies (ID, name)
- Cartoon (ID, Is_3D)
- Sci-Fi (ID, Is_Adoption)

→ A record may appear in multiple relations

→ A middle ground between OO and NULL

Many-to-one Relationship → Relation



→ Intuitive Translation:

- Products (Pname, price)
- Companies (Cname, country)
- Make (Pname, Cname) ⇒ Make (Pname, Cname)

→ Observation: each Pname has only 1 Cname

→ Simplification: Merge "Make" and "Product"

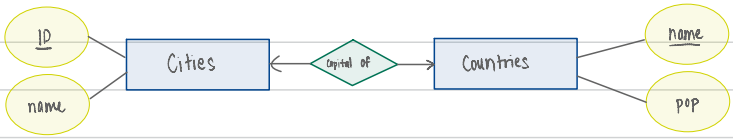
→ Results:

- Products (Pname, Price, Cname)
- Companies (Cname, country)

→ In general, do not need to create a relation for a many-to-one relationship

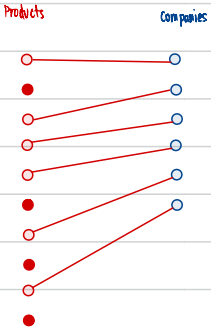
→ Just need to put key of "one" side into relation of "many" side

One-to-One Relationship → Relation



Solution 1	Solution 2
Cities (CityID, Cityname)	Cities (CityID, Cityname, Countryname)
Countries (Countryname, pop, CityID) ↓ will not be null due to referential Integrity	Countries (Countryname, pop) ↓ may be null

∴ Solution 1 is preferred.



"A city can be the capital of only 1 country"

"A country must have 1 capital"

→ no need to create a relation for a one-to-one relationship

→ Only need to put key of one side into the relation of the other