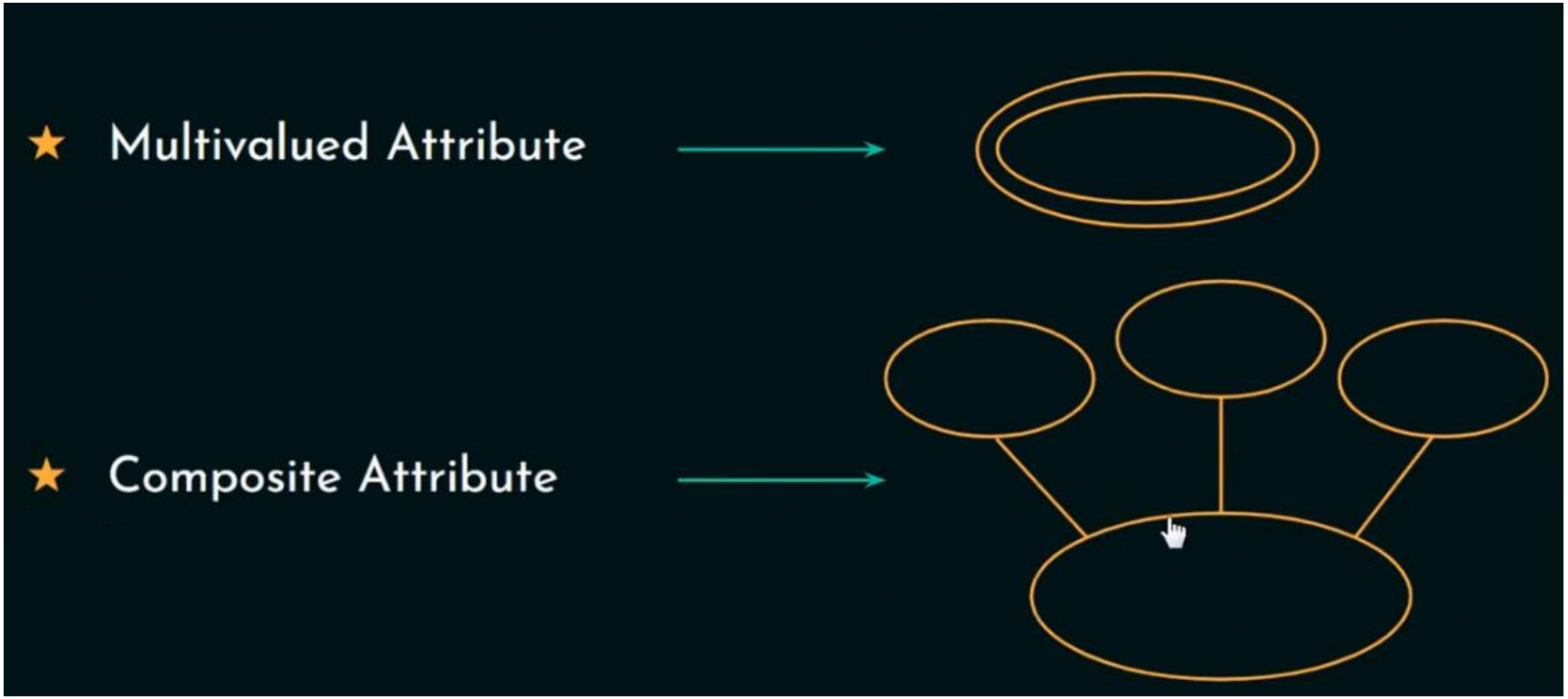
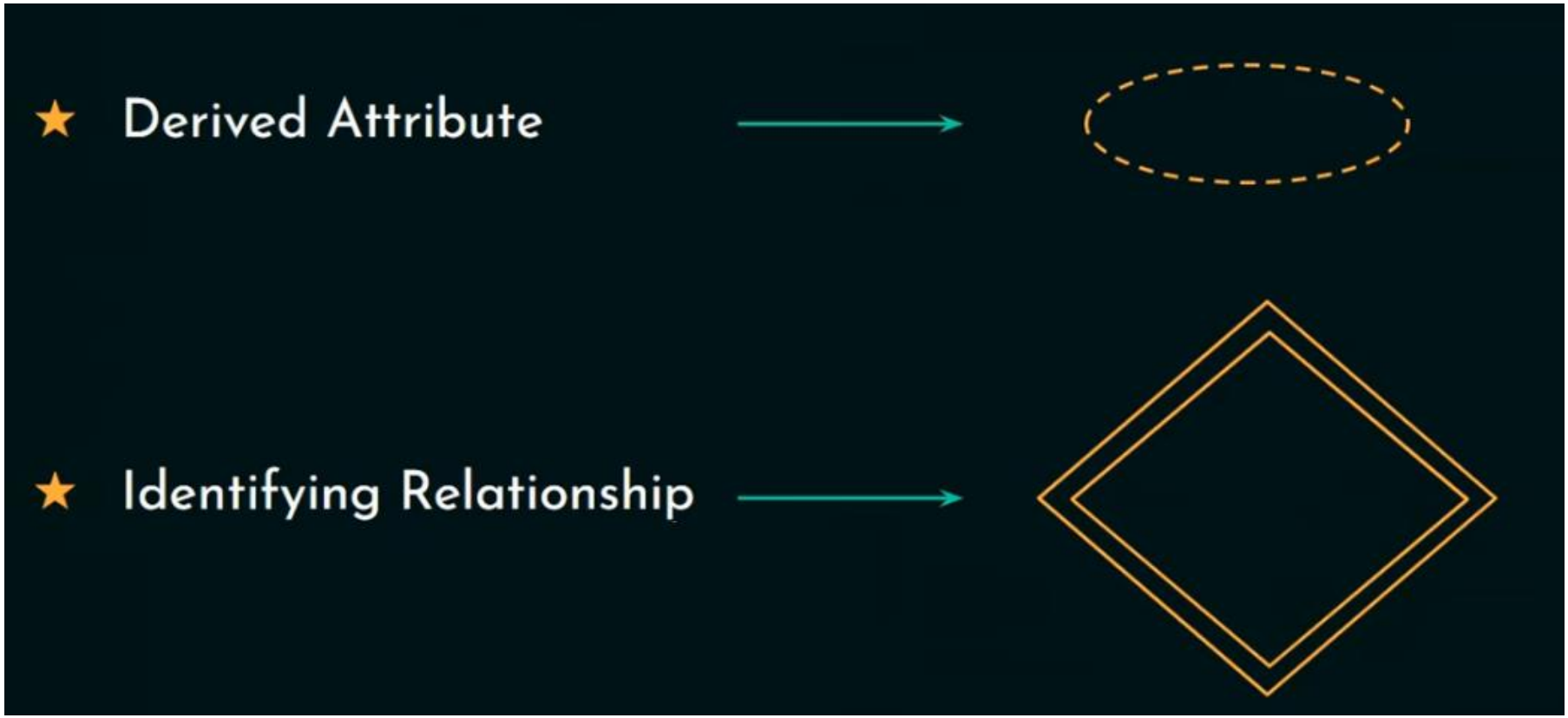
Chapter 6: Database Design Using the E-R Model (Cont.)

Database System Concepts, 7th Ed.

Extended E-R Features


Entity
Weak Entity
Attribute
Key Attribute


Multivalued Attribute
Composite Attribute


Derived Attribute
Identifying Relationship

Extended Entity Relationship (ER) Features

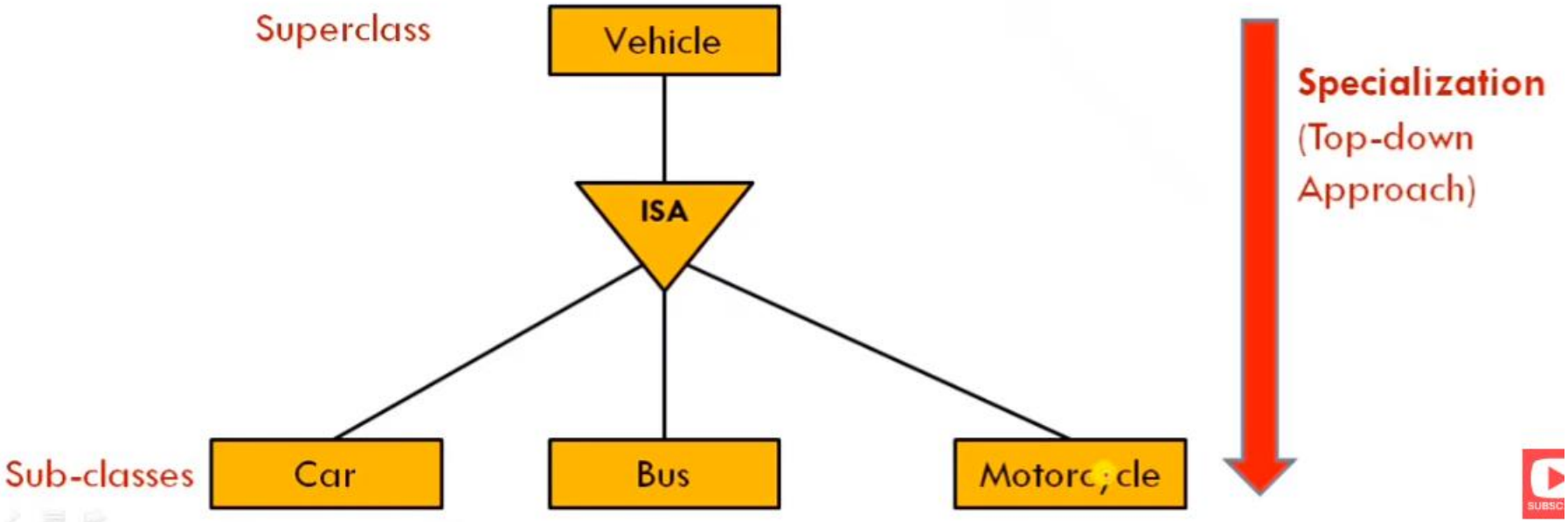
- As the complexity of data increased in the late 1980s, it became more and more difficult to use the traditional ER Model for database modelling. Hence some improvements or enhancements were made to the existing ER Model to make it able to handle the complex applications better.
- Hence, as part of the **Extended ER Model**, along with other improvements, three new concepts were added to the existing ER Model:
  1. **Generalization**
  2. **Specialization**
  3. **Aggregation**

Specialization

- Specialization is **opposite** of Generalization
- In **Specialization**, **an entity is broken down into sub-entities based on their characteristics.**
- **Specialization is a "Top-down approach"** where higher level entity is specialized into two or more lower level entities.
- **Specialization is used** to identify the subset of an entity set that shares some distinguishing characteristics.
- **Specialization** can be repeatedly applied to refine the design of schema
- depicted by triangle component labeled **ISA**

Example: Specialization

- **Vehicle** entity can be a Car, Truck or Motorcycle.
  - Normally, the superclass is defined first, the subclass and its related attributes are defined next, and relationship set are then added.

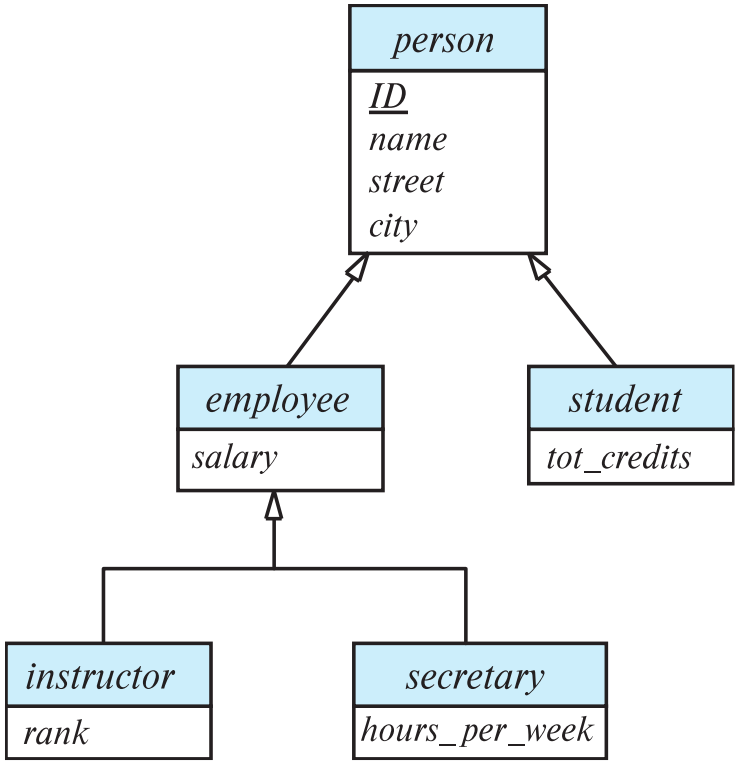


Specialization

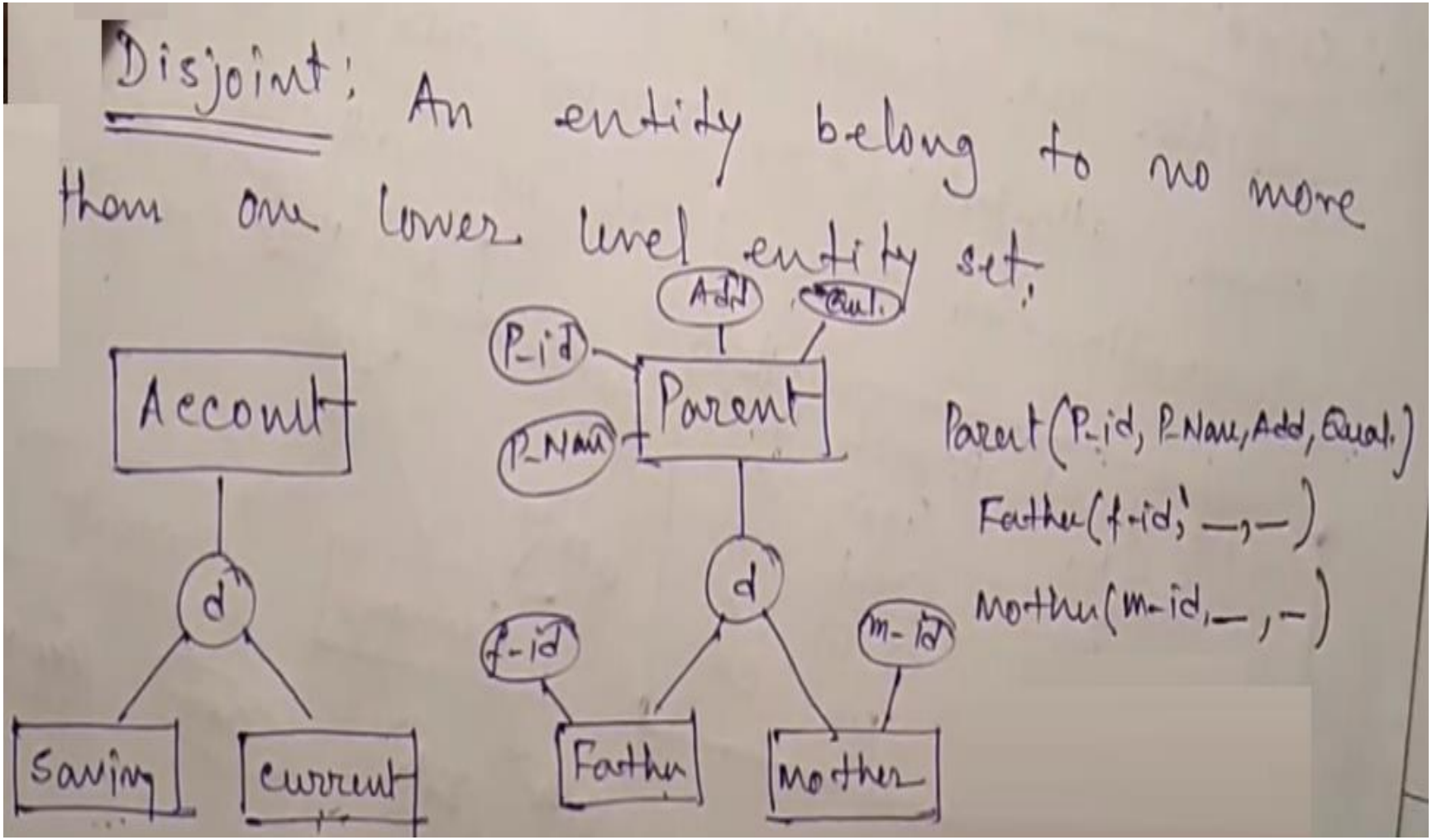
- Top-down design process; we designate sub-groupings within an entity set that are distinctive from other entities in the set.
- These sub-groupings become lower-level entity sets that have attributes or participate in relationships that do not apply to the higher-level entity set.
- Depicted by a *triangle* component labeled ISA (e.g., *instructor* "is a" *person*).
- **Attribute inheritance** – a lower-level entity set inherits all the attributes and relationship participation of the higher-level entity set to which it is linked.

Specialization Example

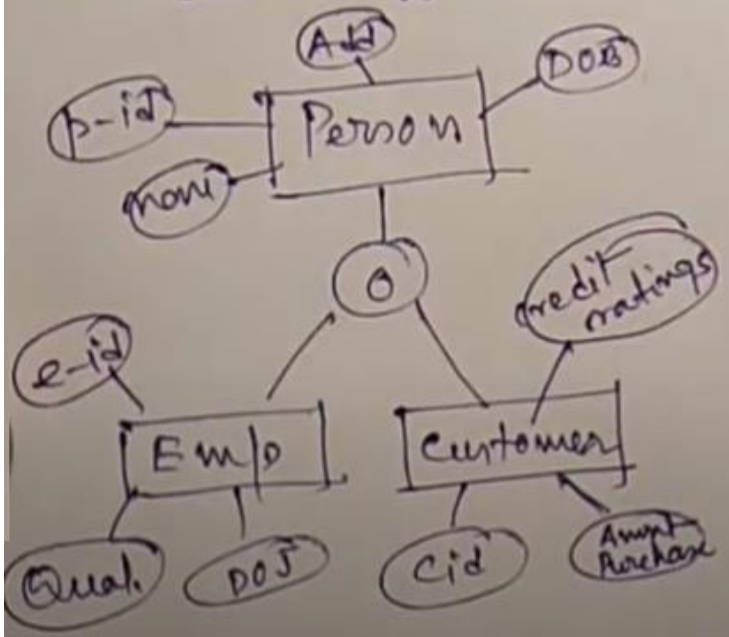
- **Overlapping** – *employee* and *student*
- **Disjoint** – *instructor* and *secretary*
- Total and partial



Disjoint: An entity belong to no more than one lower level entity set.



(b) Overlapping: The same entity may belong to more than one lower-level entity set within a single generalization.



Person (P-id, Name, Add, DOB)

EMP (e-id, Qualification, DOJ)

customer (cid, Amount Purchase, credit ratings)

Representing Specialization via Schemas

- Method 1:
  - Form a schema for the higher-level entity
  - Form a schema for each lower-level entity set, include primary key of higher-level entity set and local attributes

schema	attributes
person	ID, name, street, city
student	ID, tot_cred
employee	ID, salary

  - Drawback: getting information about, an *employee* requires accessing two relations, the one corresponding to the low-level schema and the one corresponding to the high-level schema

Representing Specialization as Schemas (Cont.)

- Method 2:
  - Form a schema for each entity set with all local and inherited attributes

schema	attributes
person	ID, name, street, city
student	ID, name, street, city, tot_cred
employee	ID, name, street, city, salary

  - Drawback: *name, street* and *city* may be stored redundantly for people who are both students and employees

Generalization

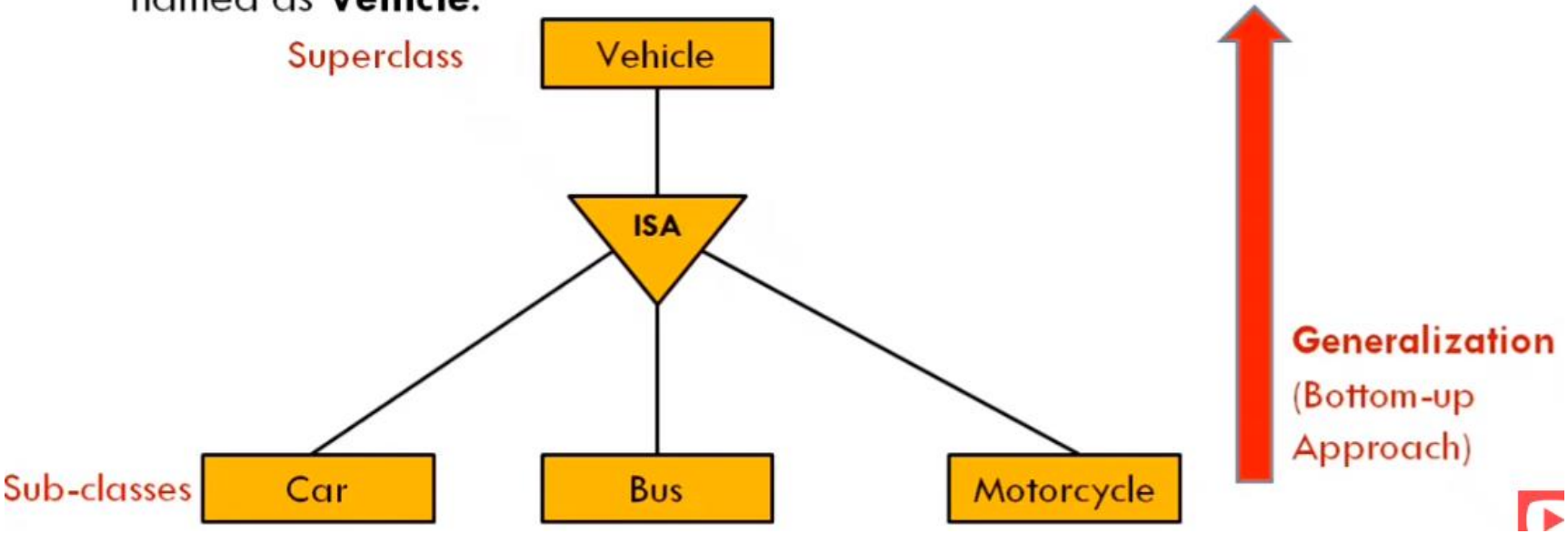
- **A bottom-up design process** – combine a number of entity sets that share the same features into a higher-level entity set.
- Specialization and generalization are simple inversions of each other; they are represented in an E-R diagram in the same way.
- The terms specialization and generalization are used interchangeably.

Generalization

- **Generalization** is the process of extracting common properties from a set of entities and create a generalized entity from it.
- **Generalization is a "bottom-up approach"** in which two or more entities can be combined to form a higher level entity if they have some attributes in common.
  - subclasses are combined to make a superclass.
- **Generalization is used** to emphasize the similarities among lower-level entity set and to hide differences in the schema

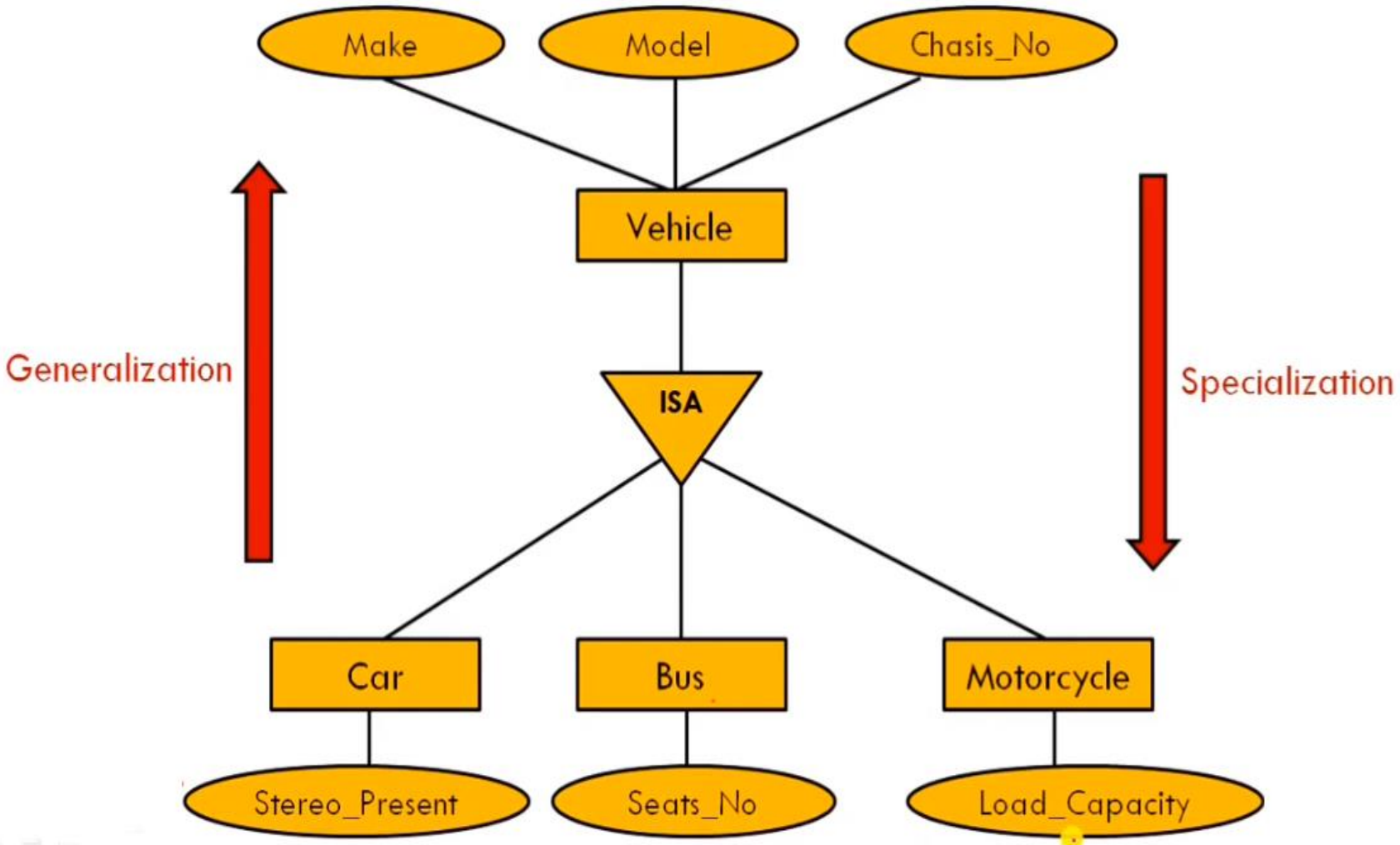
Example: Generalization

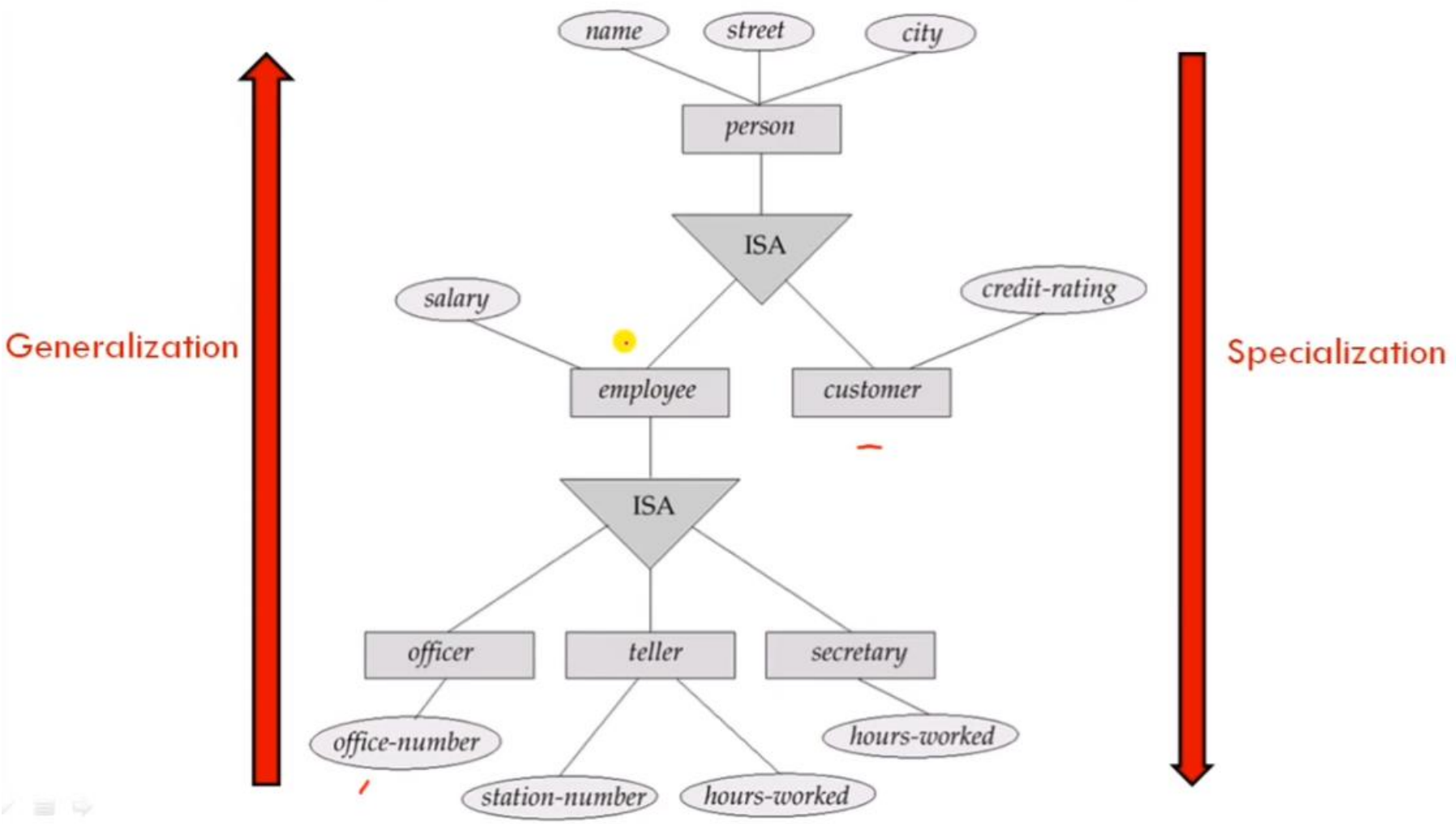
- Consider we have 3 sub entities Car, Bus and Motorcycle. Now these three entities can be generalized into one higher-level entity (or super class) named as **Vehicle.**



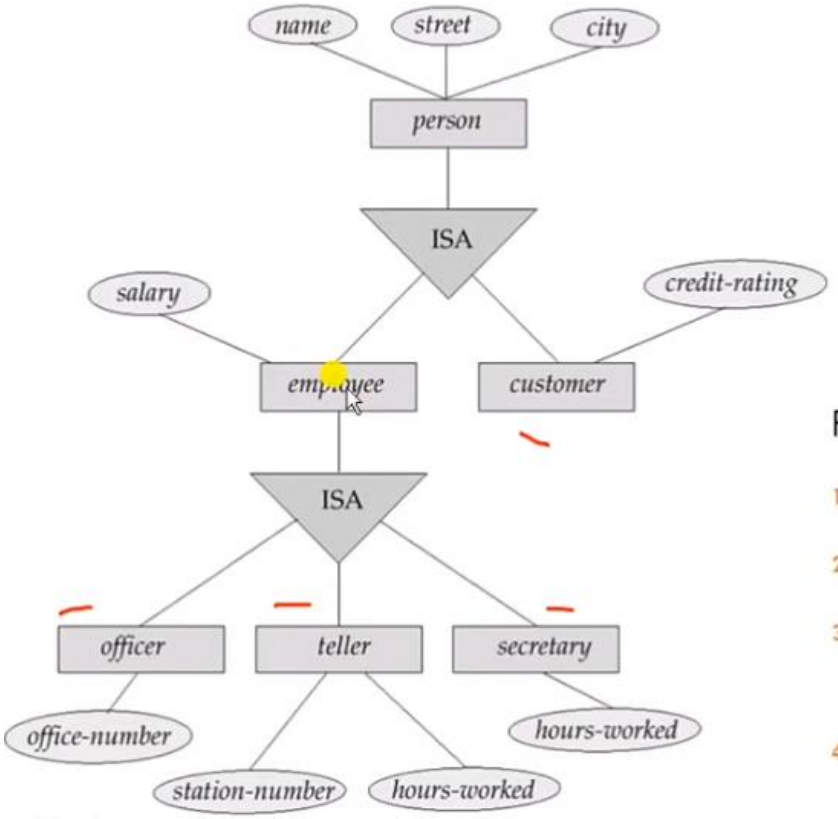
Inheritance

- **Inheritance** is an important feature of generalization and specialization.
- **Attribute inheritance allows lower level entities to *inherit the attributes* of higher level entities.**
  - **For example,** Consider relations Car and Bus inheriting the attributes of Vehicle. Thus, Car is described by attributes of super-class Vehicle as well as its own attributes.
- This also extends to **Participation Inheritance** in which ***relationships involving higher-level entity-sets are also inherited by lower-level entity-sets.***
  - A lower-level entity-set can participate in its own relationship-sets, too


Make
Model
Chasis_No
Vehicle
Generalization
ISA
Specialization
Car
Bus
Motorcycle
Stereo_Present
Seats_No
Load_Capacity


name
street
city
person
ISA
salary
credit-rating
employee
customer
ISA
officer
teller
secretary
office-number
station-number
hours-worked
hours-worked
Generalization
Specialization

How Schema or Tables can be formed?

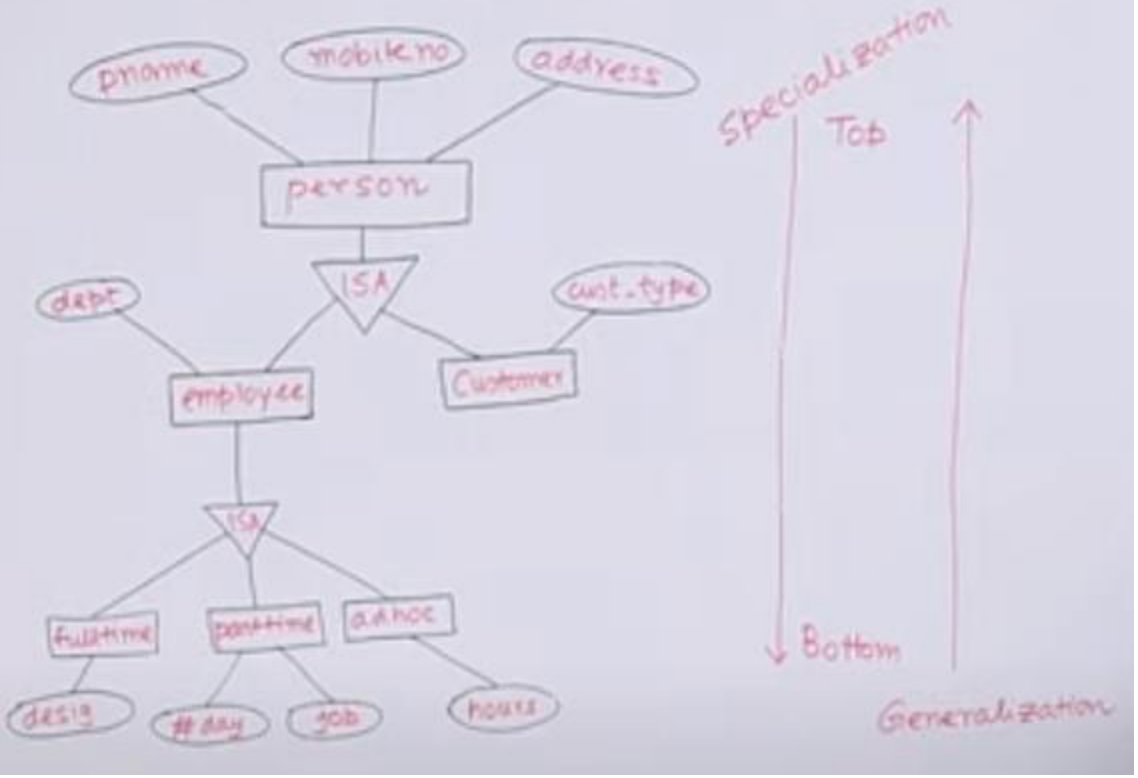


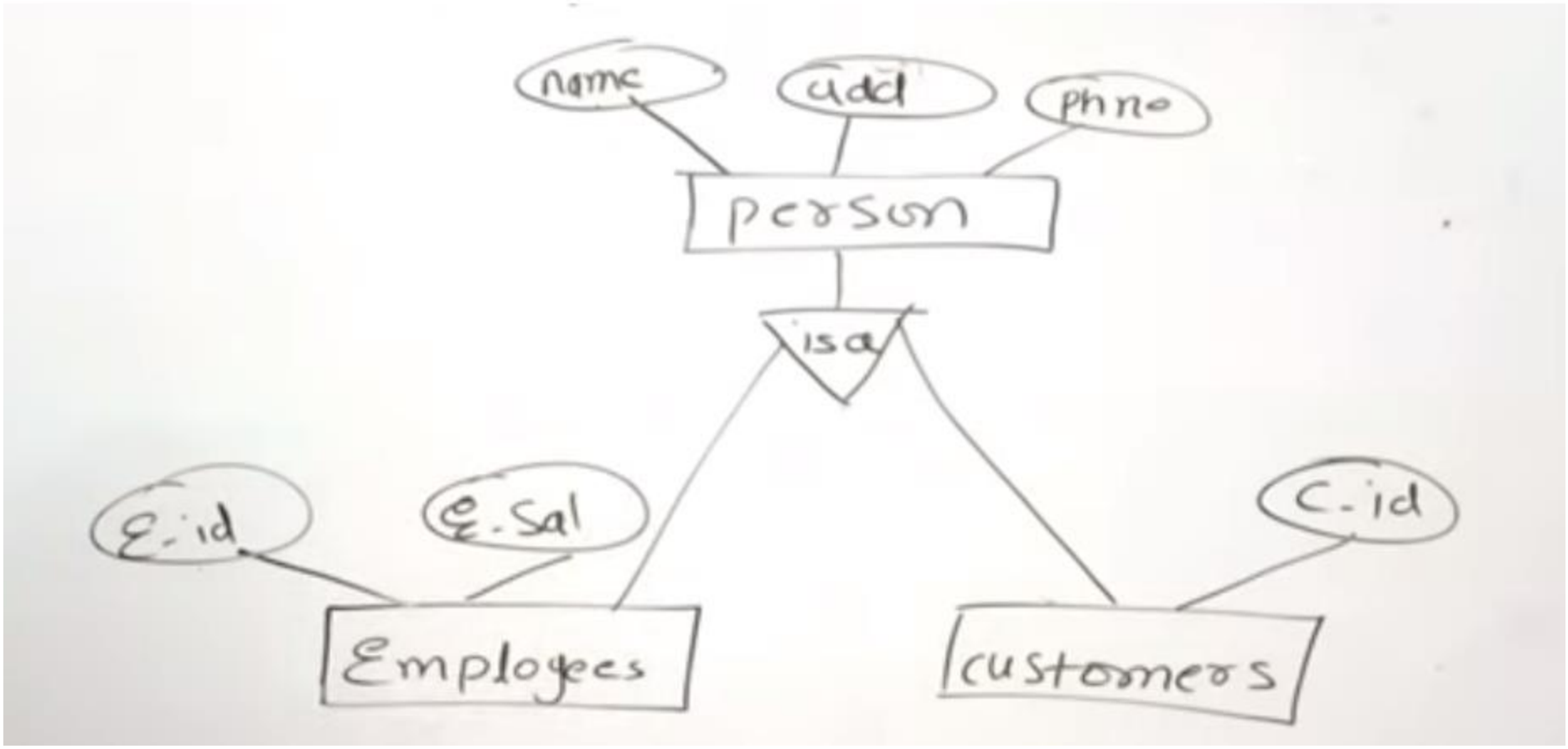
Four tables can be formed:

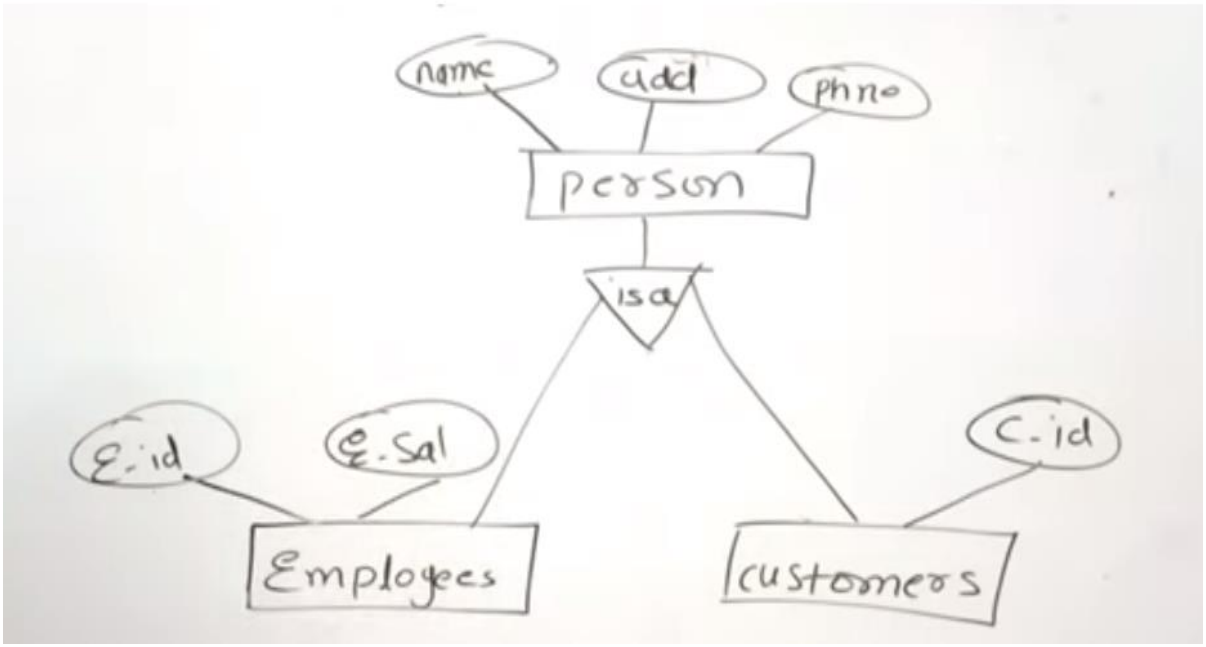
1. **customer** (name, street, city, credit_rating)
2. **officer** (name, street, city, salary, office_number)
3. **teller** (name, street, city, salary, station_number, hours_worked)
4. **secretary** (name, street, city, salary, hours_worked)

Customer(pname, mobileno, address, cust-type)
fulltime(pname, mobileno, address, dept, desig)
parttime(pname, mobileno, address, dept, #day, job)
adhoc(pname, mobileno, address, dept, hours)

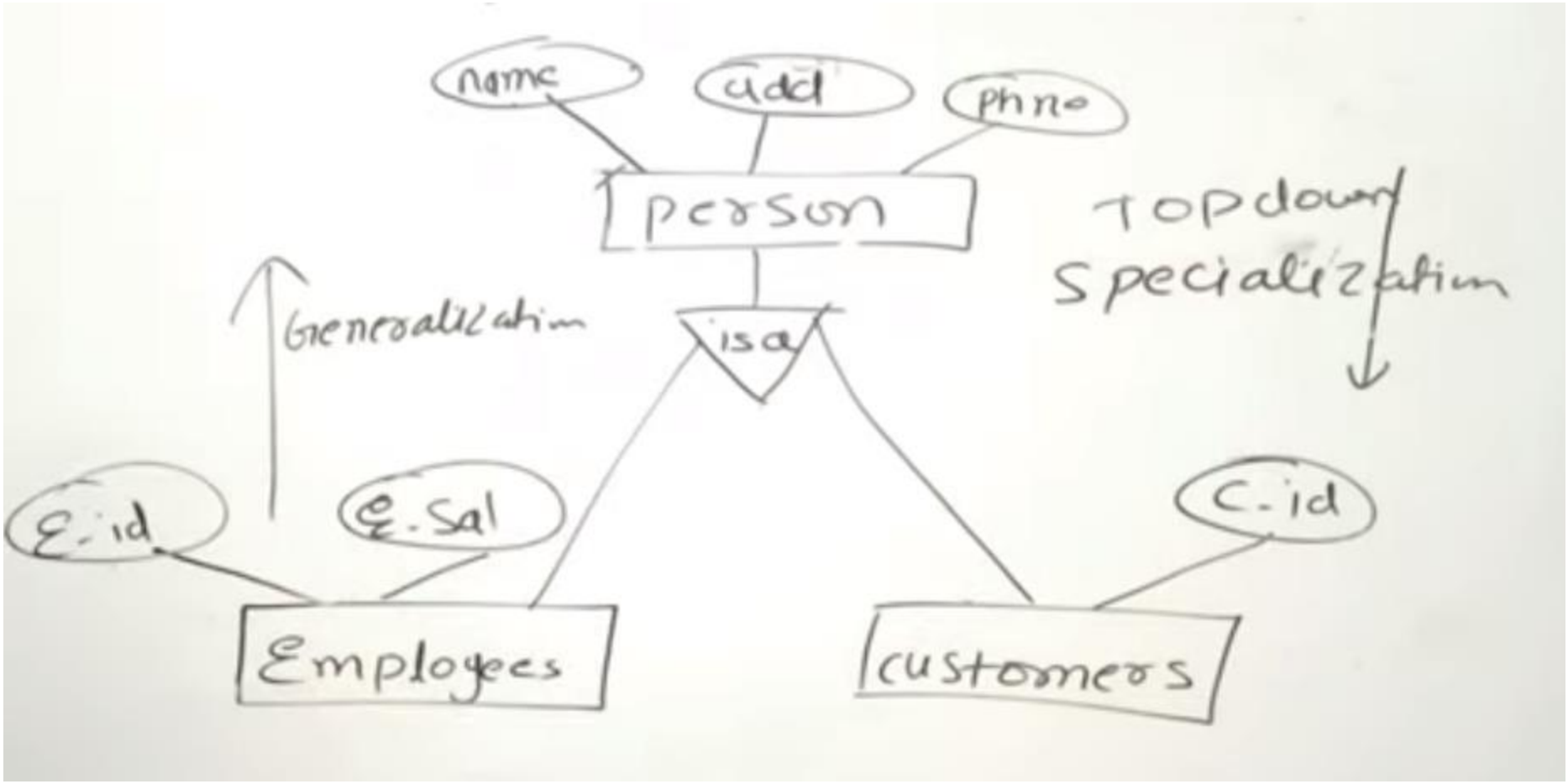
Specialization and Generalization




name
add
phno
person
isa
E.id
E.Sal
C.id
Employees
customers



employees (name add, phno, E.id E.Sal)
cutomers (name, add, Pho, C.id)
person (name add, phno)


name
add
phno
person
isa
Generalization
Top down
Specialization
E.id
E.Sal
C.id
Employees
customers

Completeness constraint

- **Completeness constraint** -- specifies whether or not an entity in the higher-level entity set must belong to at least one of the lower-level entity sets within a generalization.
  - **total**: an entity must belong to one of the lower-level entity sets
  - **partial**: an entity need not belong to one of the lower-level entity sets

Completeness constraint (Cont.)

- Partial generalization is the default.
- We can specify total generalization in an ER diagram by adding the keyword **total** in the diagram and drawing a dashed line from the keyword to the corresponding hollow arrow-head to which it applies (for a total generalization), or to the set of hollow arrow-heads to which it applies (for an overlapping generalization).
- The *student* generalization is total: All student entities must be either graduate or undergraduate. Because the higher-level entity set arrived at through generalization is generally composed of only those entities in the lower-level entity sets, the completeness constraint for a generalized higher-level entity set is usually total

Constraints in Supertype/ Completeness Constraint

- ***Completeness Constraints***: Whether an instance of a supertype ***must*** also be a member of at least one subtype
  - Total Specialization Rule: Yes (double line)
  - Partial Specialization Rule: No (single line)

Figure 3-6 Examples of completeness constraints

a) Total specialization rule

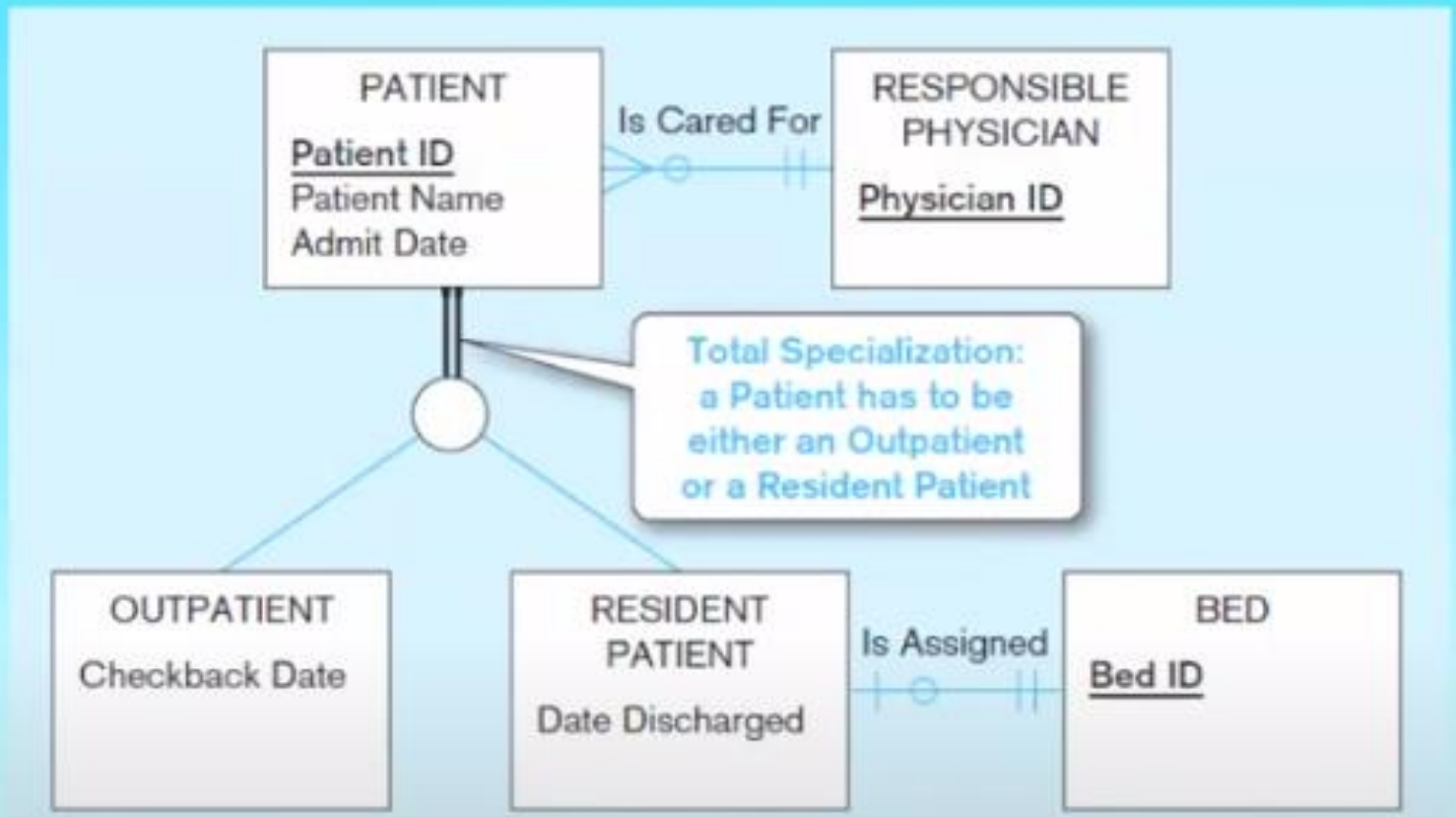
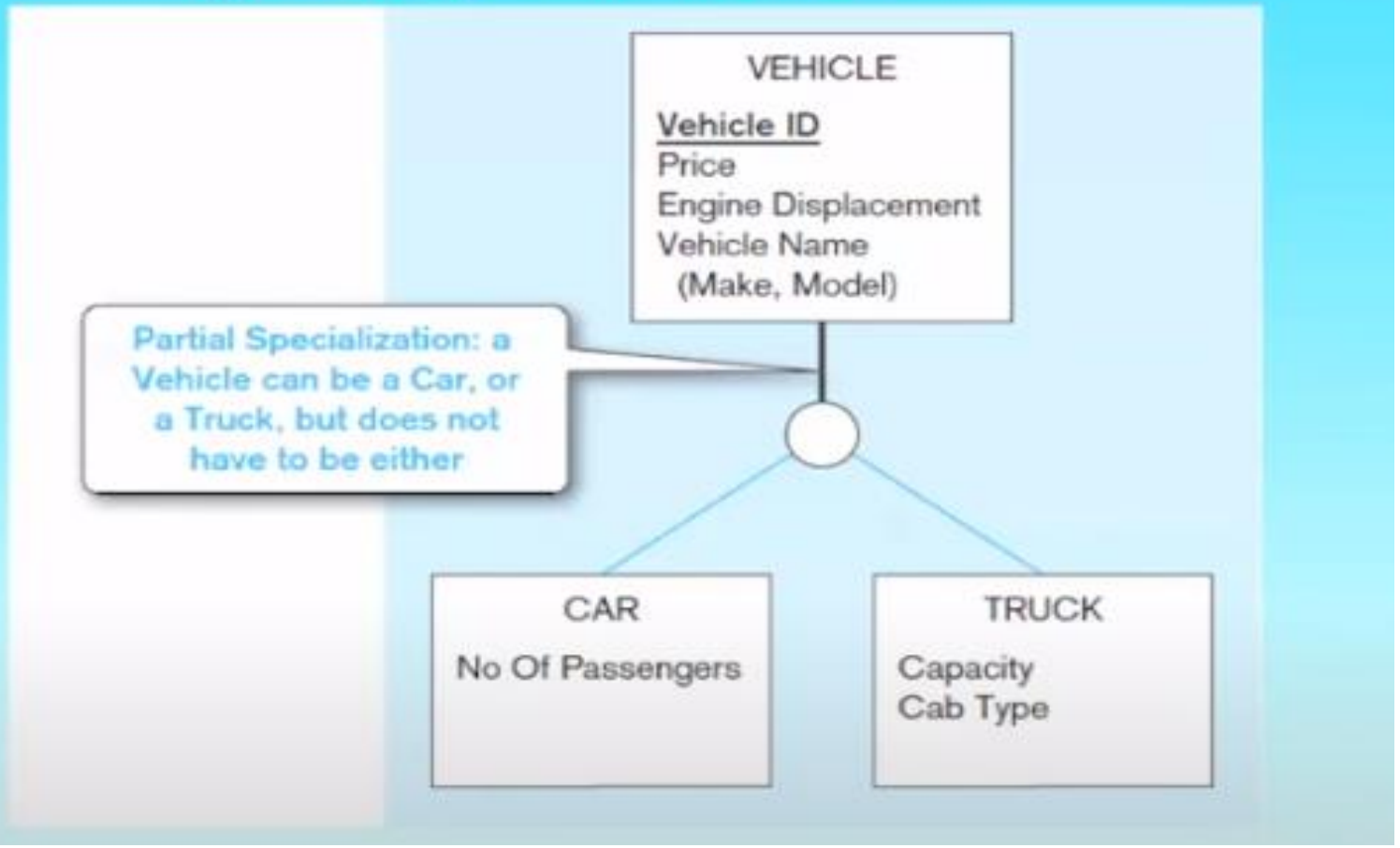


Figure 3-6 Examples of completeness constraints (cont.)

b) Partial specialization rule

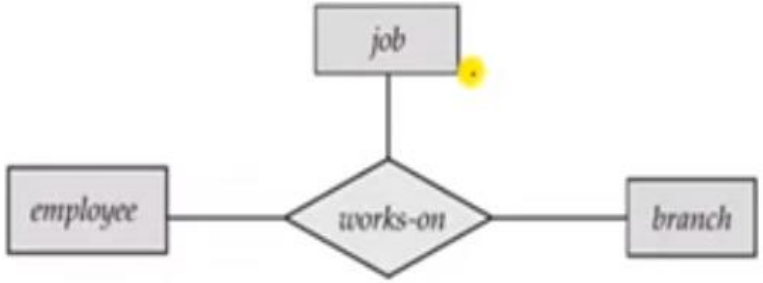


Aggregation

- **Aggregation** is used when we need to express a **relationship among relationships**
- **Aggregation** is an **abstraction** through which **relationships are treated as higher level entities**
- **Aggregation** is a process when a **relationship between two entities is considered as a single entity** and again this single entity has a **relationship** with **another entity**

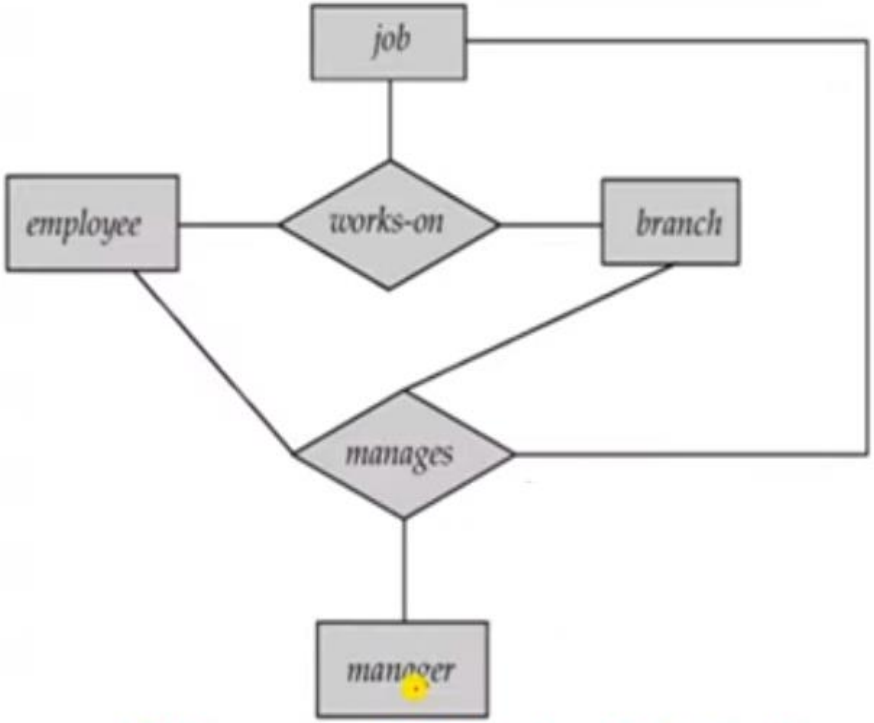
Example: (Relationship of Relations)

- Basic E-R model can't represent relationships involving other relationships
- Consider a ternary relationship **works_on** between **Employee, Branch and Job.**
  - Am **employee** ***works on*** a particular **job** at a particular **branch**



- Suppose we want to assign a **manager** for jobs performed by an employee at a branch (i.e. want to assign managers to each employee, job, branch combination)
  - Need a separate manager entity-set
  - Relationship between each manager, employee, branch, and job entity

Example: (Redundant Relationship)



ER Diagram with Redundant Relationship

- Relationship sets ***works-on*** and ***manages*** represent overlapping (redundant) information
  - Every *manages* relationship corresponds to a *works-on* relationship
  - However, some *works-on* relationships may not correspond to any *manages* relationships
    - So we can't discard the *works-on* relationship

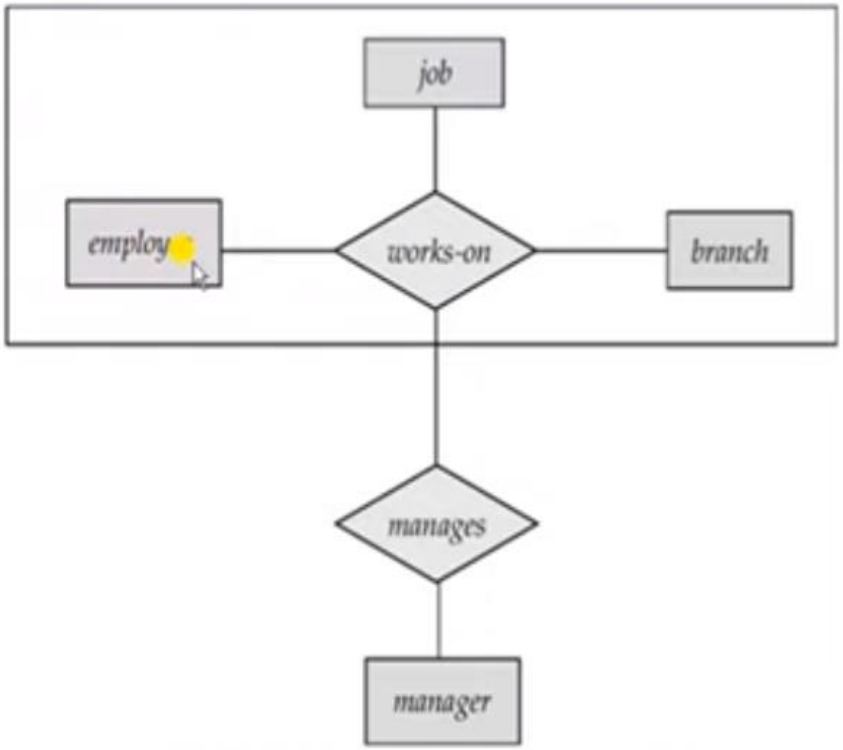
Aggregation

- Eliminate this redundancy via ***aggregation***
  - Treat relationship as an abstract entity
  - Allows relationships between relationships
  - Abstraction of relationship into new entity

Example: (Aggregation)

With **Aggregation** (without introducing redundancy) the ER diagram can be represented as:

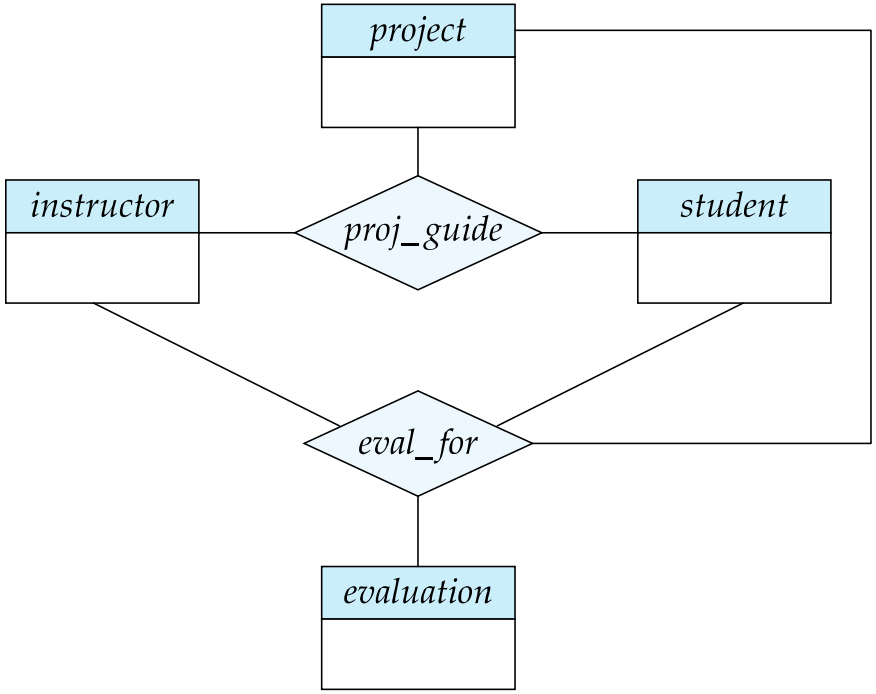
- An employee works on a particular job at a particular branch
- An employee, branch, job combination may have an associated manager



ER Diagram with Aggregation

Aggregation

- Consider the ternary relationship *proj_guide*, which we saw earlier
- Suppose we want to record evaluations of a student by a guide on a project

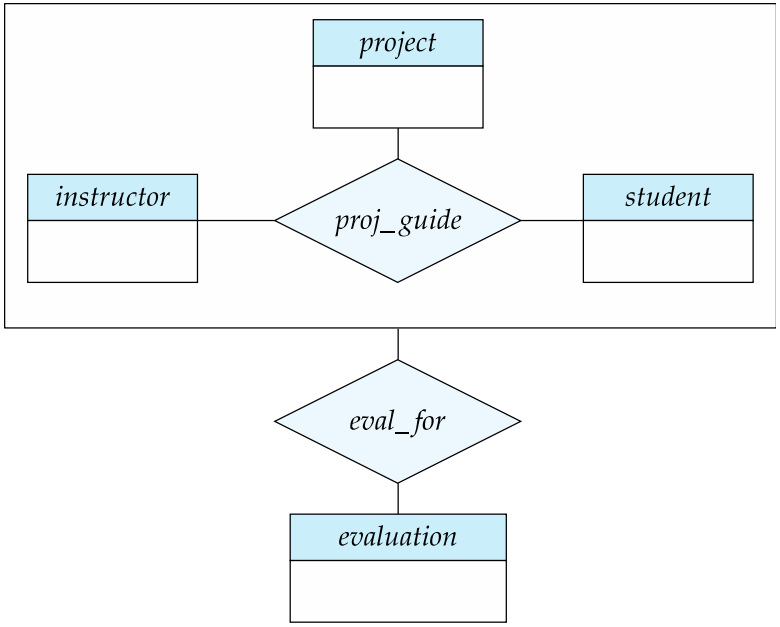


Aggregation (Cont.)

- Relationship sets *eval_for* and *proj_guide* represent overlapping information
  - Every *eval_for* relationship corresponds to a *proj_guide* relationship
  - However, some *proj_guide* relationships may not correspond to any *eval_for* relationships
    - So we can't discard the *proj_guide* relationship
- Eliminate this redundancy via *aggregation*
  - Treat relationship as an abstract entity
  - Allows relationships between relationships
  - Abstraction of relationship into new entity

Aggregation (Cont.)

- Eliminate this redundancy via *aggregation* without introducing redundancy, the following diagram represents:
  - A student is guided by a particular instructor on a particular project
  - A student, instructor, project combination may have an associated evaluation

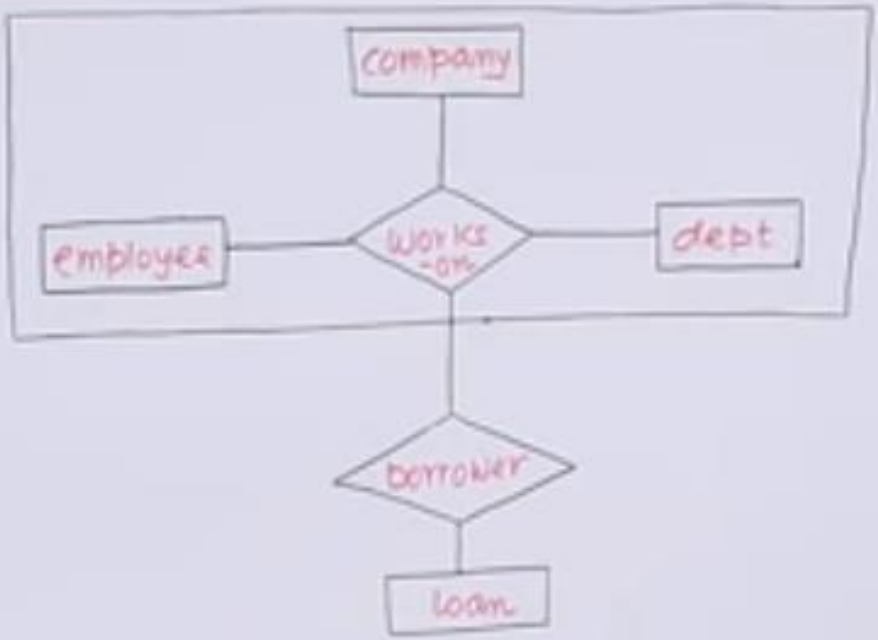


Reduction to Relational Schemas

- To represent aggregation, create a schema containing
  - Primary key of the aggregated relationship,
  - The primary key of the associated entity set
  - Any descriptive attributes
- In our example:
  - The schema *eval_for* is:

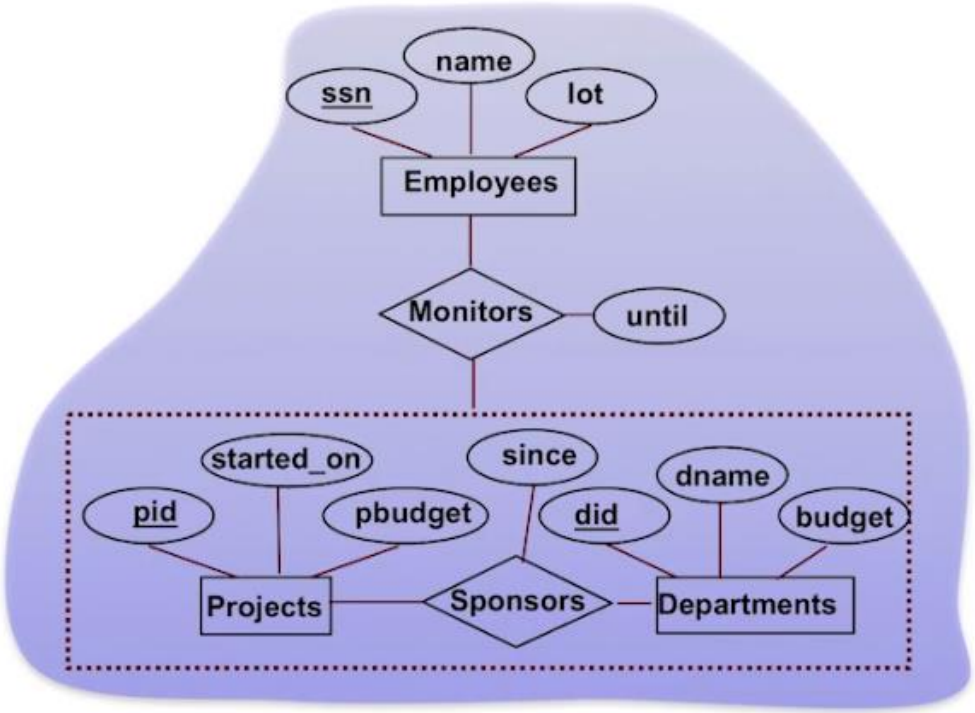
    eval_for (*s_ID, project_id, i_ID, evaluation_id*)
  - The schema *proj_guide* is redundant.

ER Diagram with aggregation

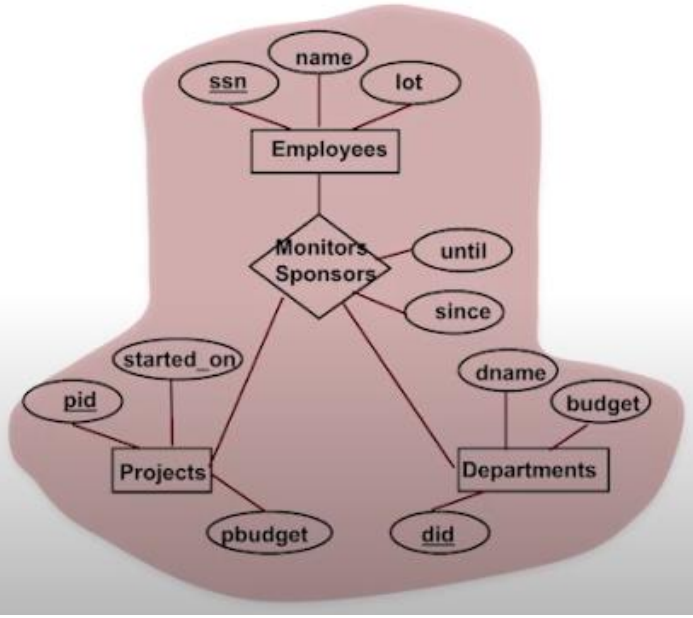


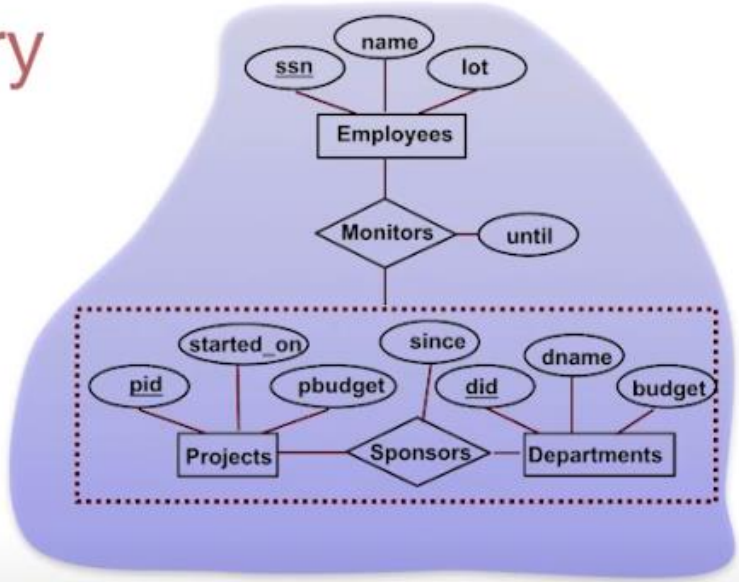
Aggregation

Allows relationships to have relationships.

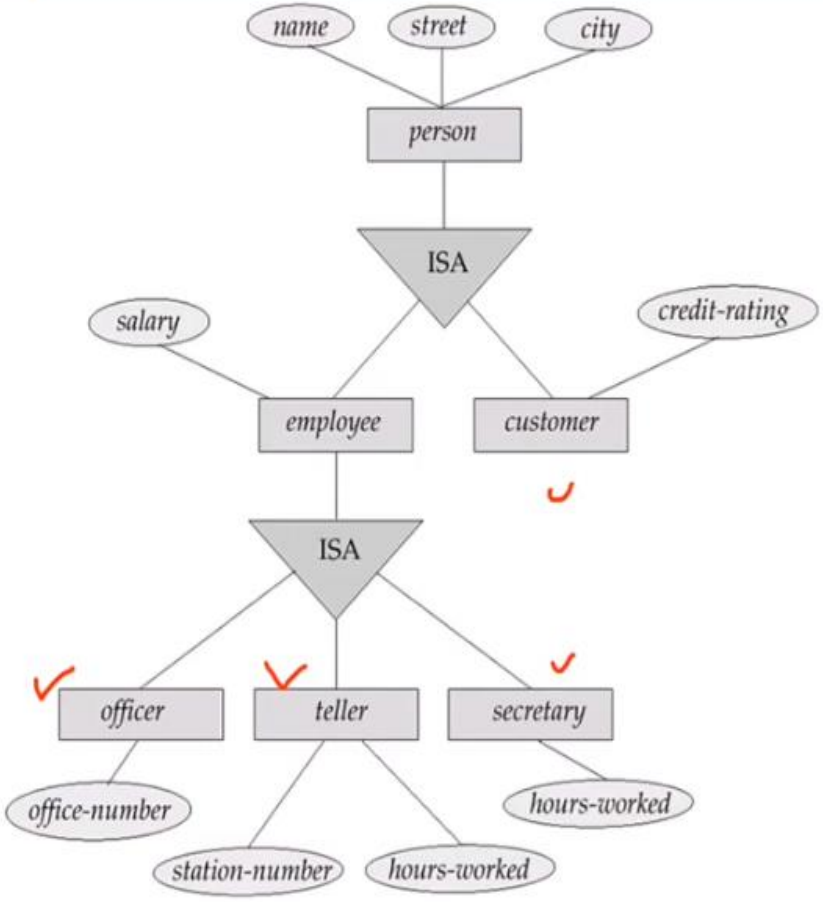


Aggregation vs. Ternary





Question 4:

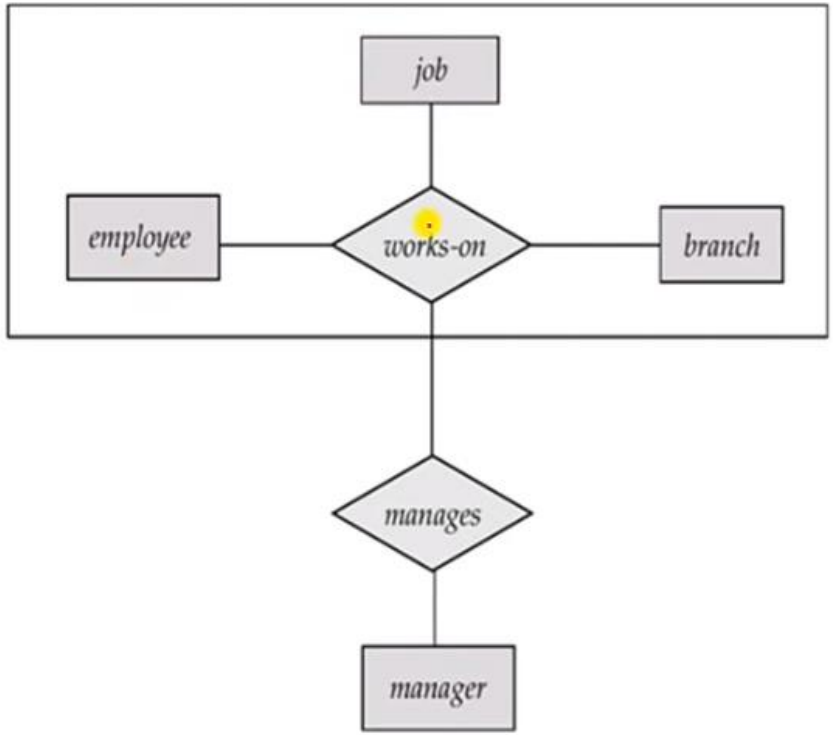


Solution-

Minimum 4 tables will be required:

1. **customer** (name, street, city, credit_rating)
2. **officer** (name, street, city, salary, office_number)
3. **teller** (name, street, city, salary, station_number, hours_worked)
4. **secretary** (name, street, city, salary, hours_worked)

Question 5:



ER Diagram with Aggregation

Solution:

Minimum six tables will be required

Job schemas:

1. employee
2. job
3. Branch
4. works_on

Manager schemas:

5. manager
6. manages

We will discuss the basic design issues of ER database schema in the following points.

- **Entity vs Attribute**
- **Entity vs Relationship**
- **Binary vs Ternary Relationship**

Use of Entity Set Vs Attributes

- Choice mainly depends on the structure of the enterprise being modeled, and on the semantics associated with the attribute in question

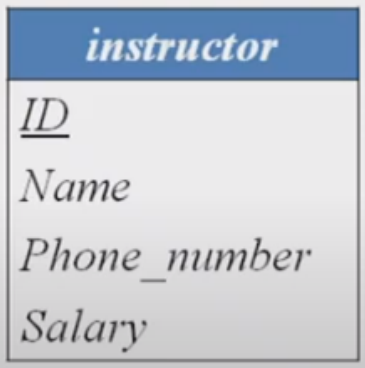
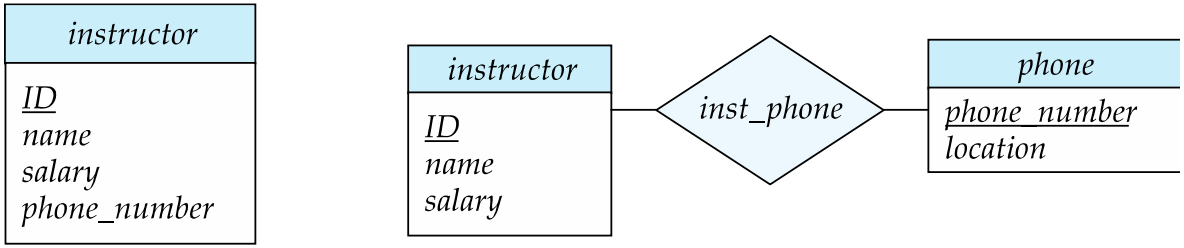


FIG A

FIG B

Entities vs. Attributes

- Use of entity sets vs. attributes



- Use of phone as an entity allows extra information about phone numbers (plus multiple phone numbers)

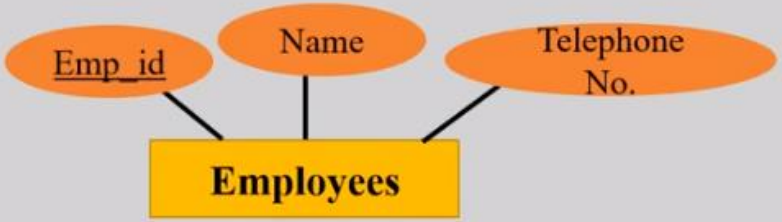
Use of Entity Sets Vs Relationship Sets

FIG A

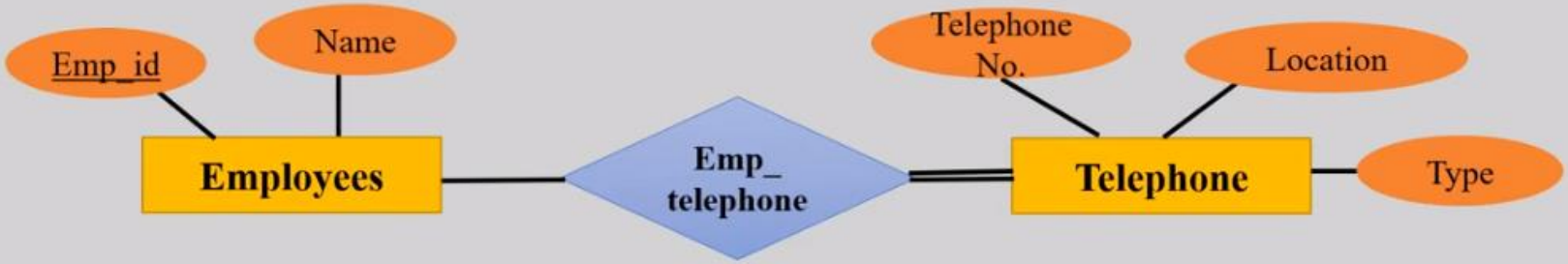
FIG B

Entity vs Attribute

Is Telephone-No an attribute or an entity set?



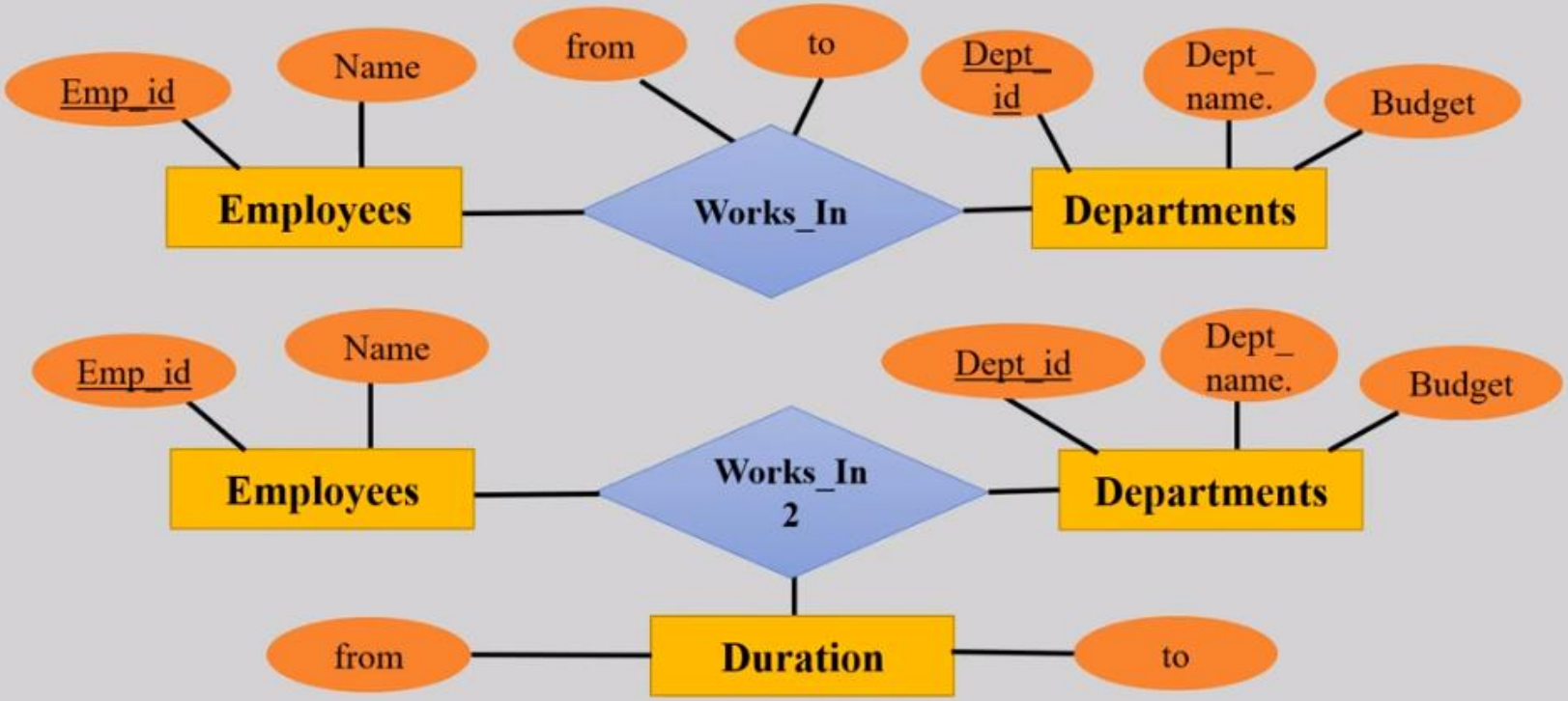
Using attribute as Entity



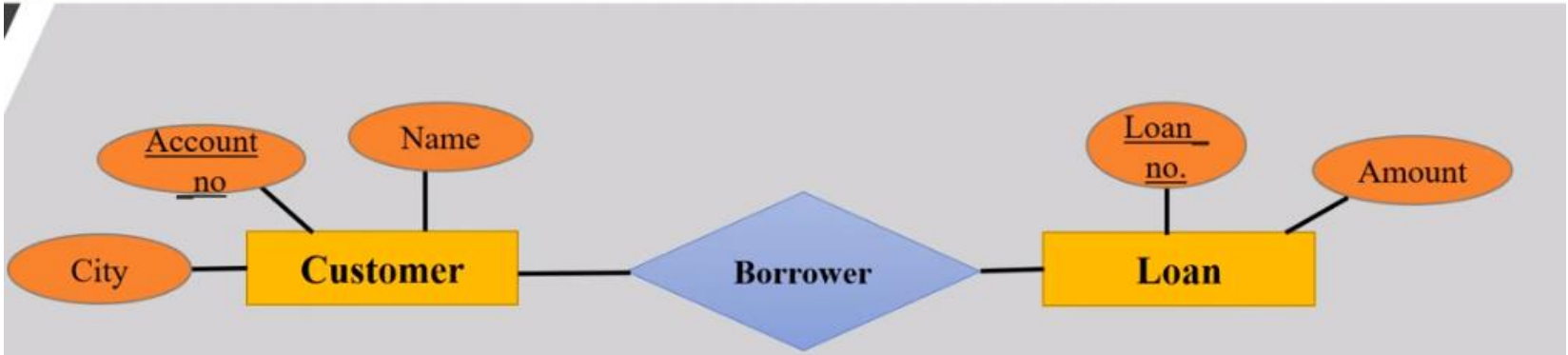
Use of telephone as an entity allows extra information about phone numbers (plus multiple phone numbers.)

Entity vs Attribute

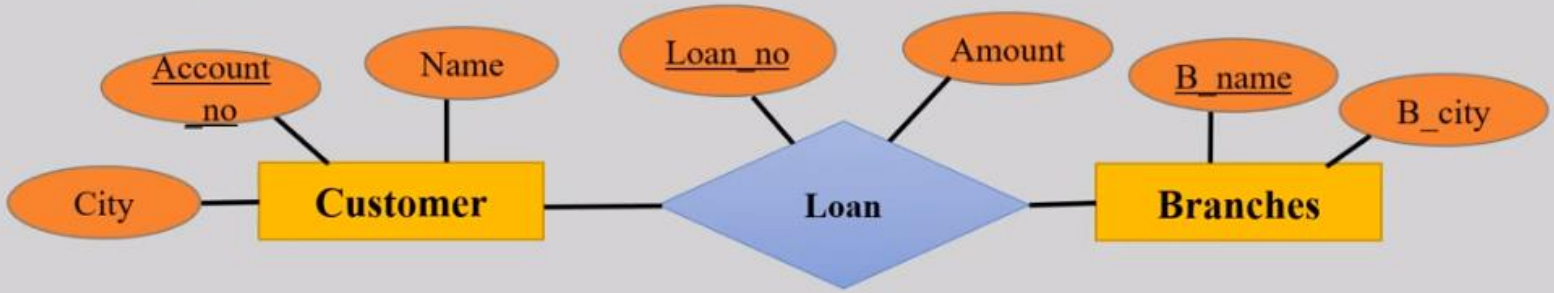
Works_In does not allow an employee to work in a department for two or more periods



Entity vs Relationship



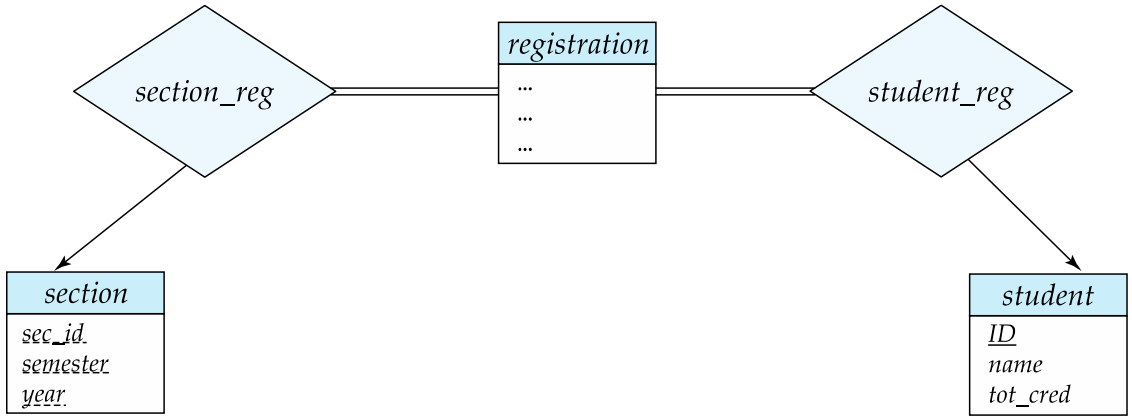
An alternative is to model a loan not as an entity, but rather as a relationship between ***customer*** and ***branches***



Entities vs. Relationship sets

- **Use of entity sets vs. relationship sets**

  Possible guideline is to designate a relationship set to describe an action that occurs between entities



- **Placement of relationship attributes**

  For example, attribute date as attribute of advisor or as attribute of student

Binary Vs *n-array* Relationship Sets

- It is always possible to replace a non-binary relationship set by a number of distinct binary relationship set

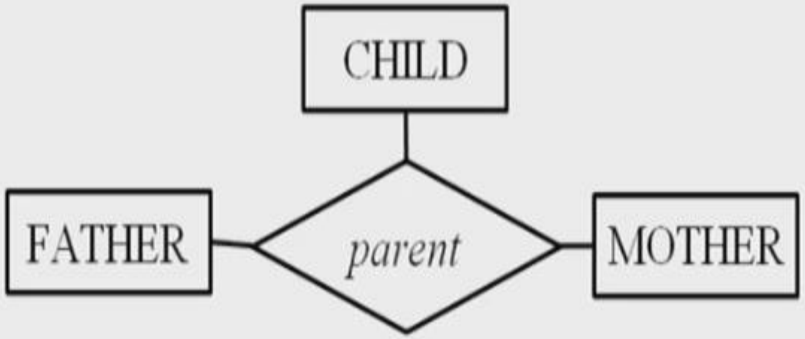


FIG A

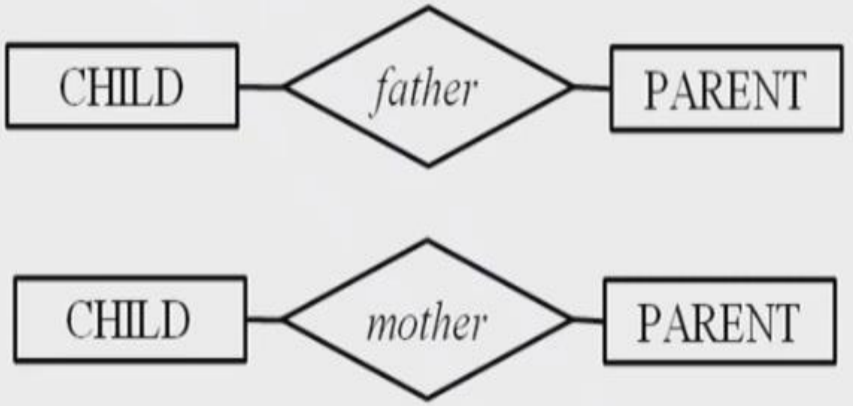
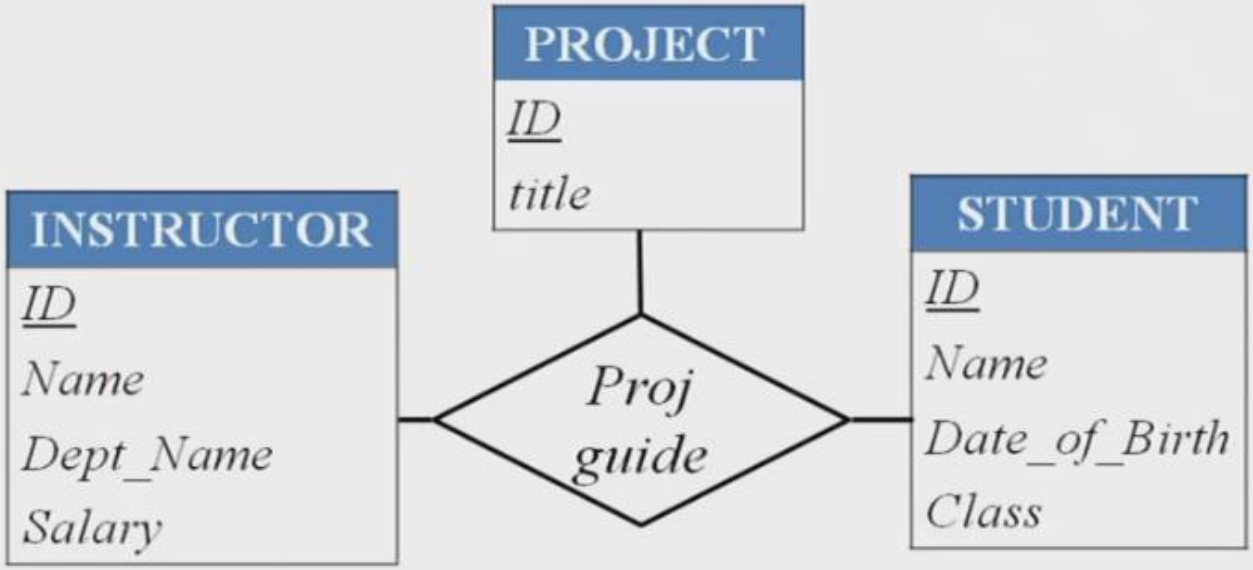


FIG B

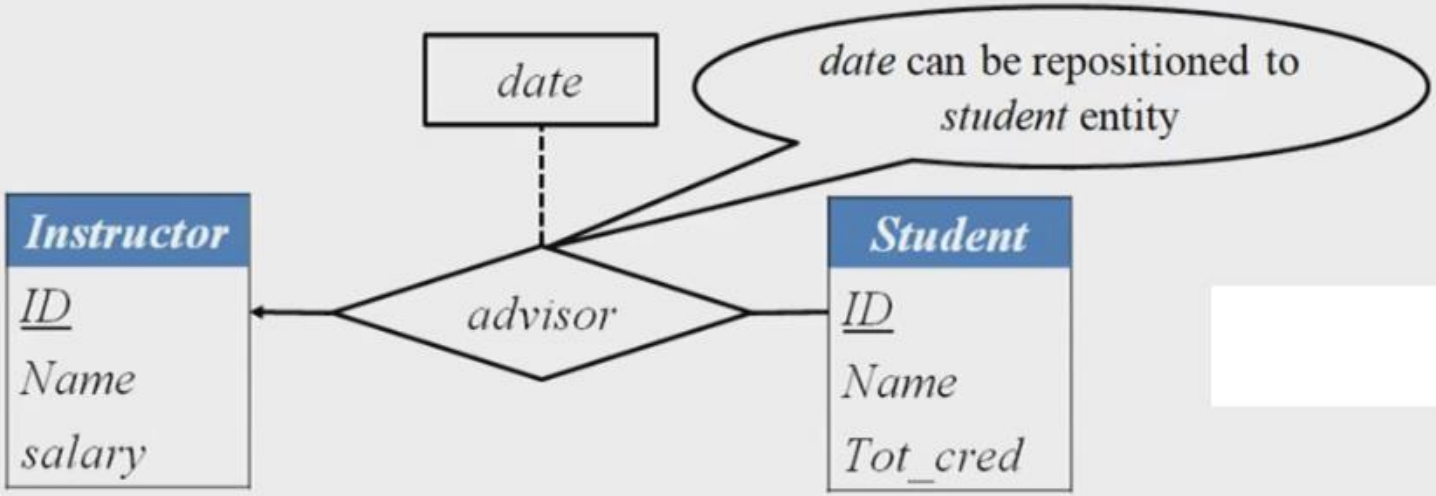
Binary Vs *n-array* Relationship Sets

- There are some relationship that are naturally non-binary
- Eg: *proj_guide*



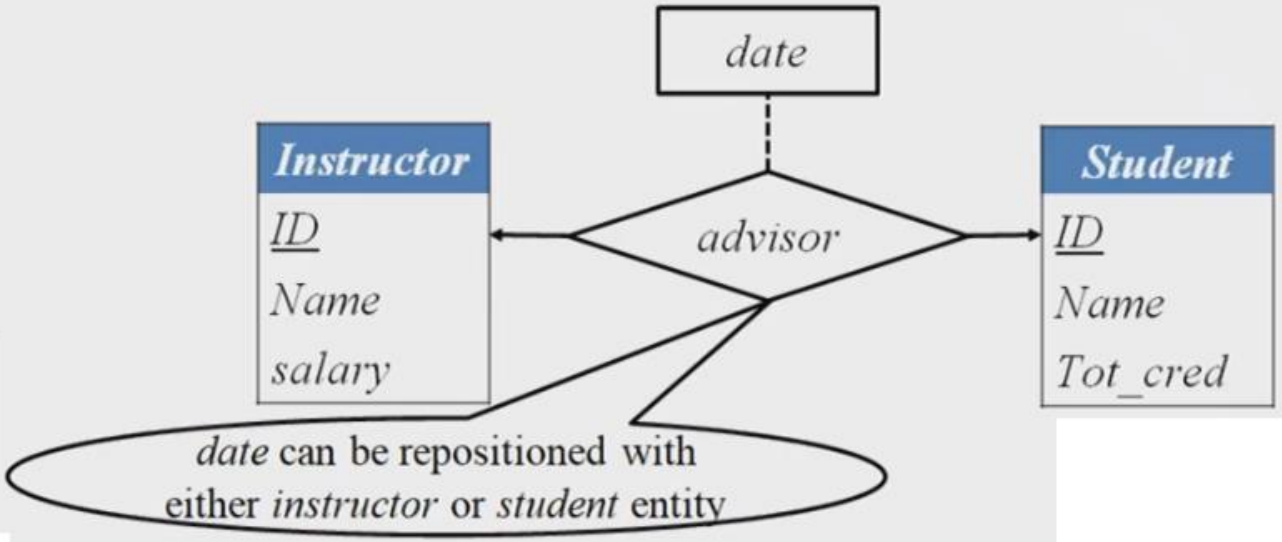
Placement of Relationship Attributes

- Attributes of one – to – many relationship sets can be repositioned to only the entity set on the many side of the relationship.



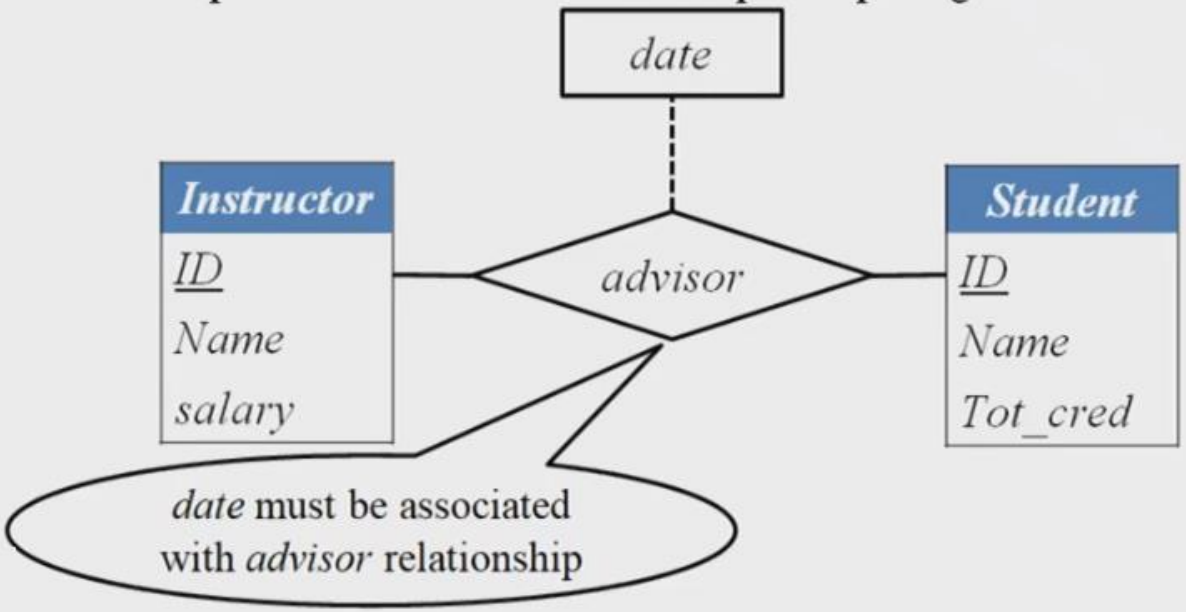
Placement of Relationship Attributes

- For one – to – one relationship sets, the relationship attribute can be associated with either one of the participating entities.



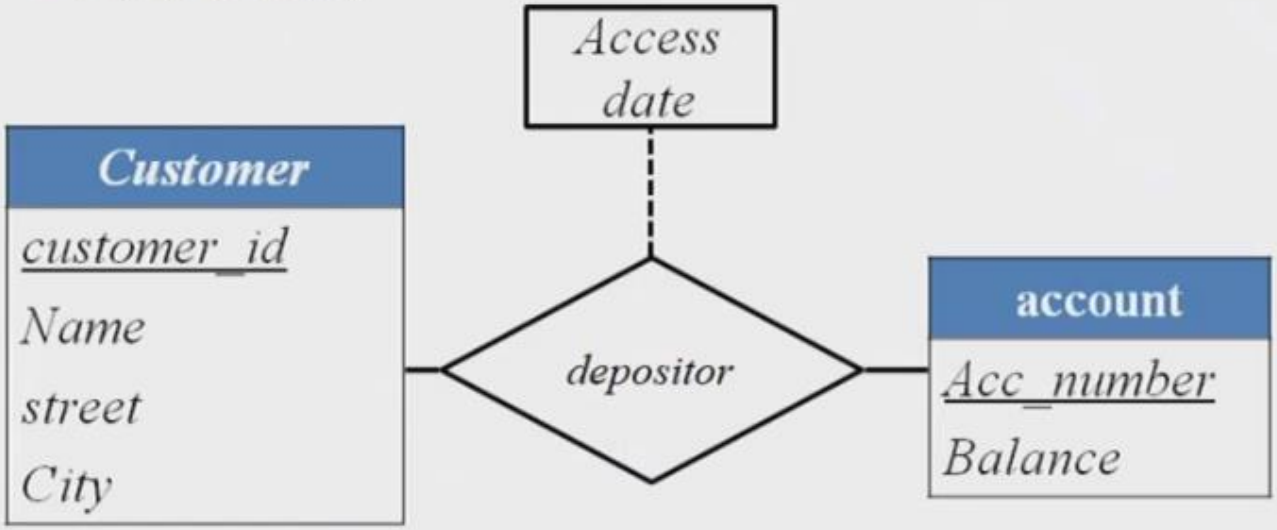
Placement of Relationship Attributes

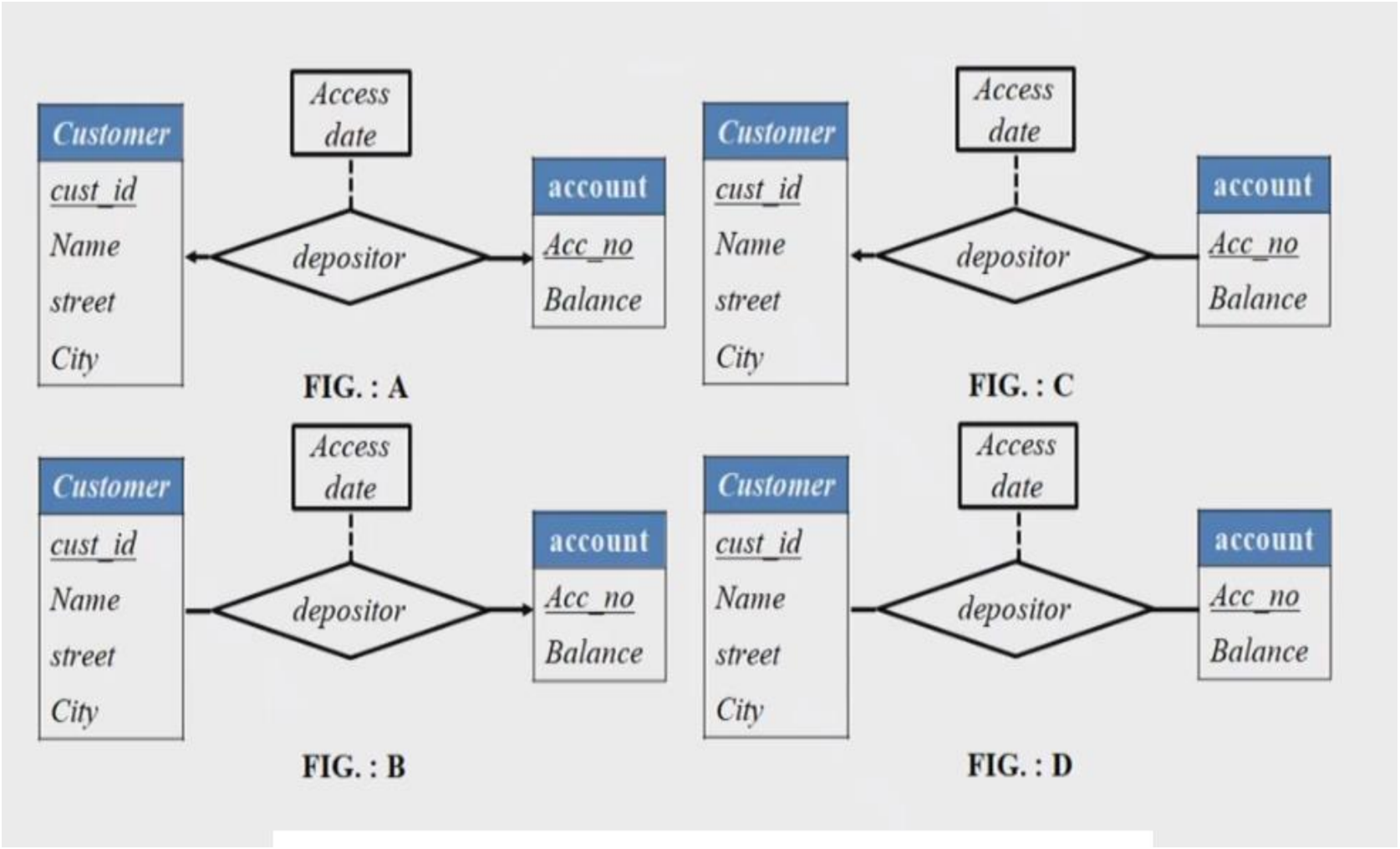
- For many – to – many relationship set, attributes must be associated with relationship sets rather than one of the participating entities.



THINK AND WRITE

- For the following ER Diagram specify the attribute placement of relationship
  - One-to-one
  - One-to-many
  - Many-to-many




Customer
cust_id
Name
street
City
Access date
depositor
account
Acc_no
Balance
FIG. : A
FIG. : C
FIG. : B
FIG. : D

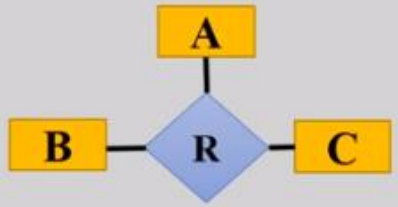
Binary Vs. Non-Binary Relationships

- Although it is possible to replace any non-binary (*n*-ary, for *n* > 2) relationship set by a number of distinct binary relationship sets, a *n*-ary relationship set shows more clearly that several entities participate in a single relationship.
- Some relationships that appear to be non-binary may be better represented using binary relationships
  - For example, a ternary relationship *parents*, relating a child to his/her father and mother, is best replaced by two binary relationships, *father* and *mother*
    - Using two binary relationships allows partial information (e.g., only mother being known)
  - But there are some relationships that are naturally non-binary
    - Example: *proj_guide*

Binary vs Ternary Relationship

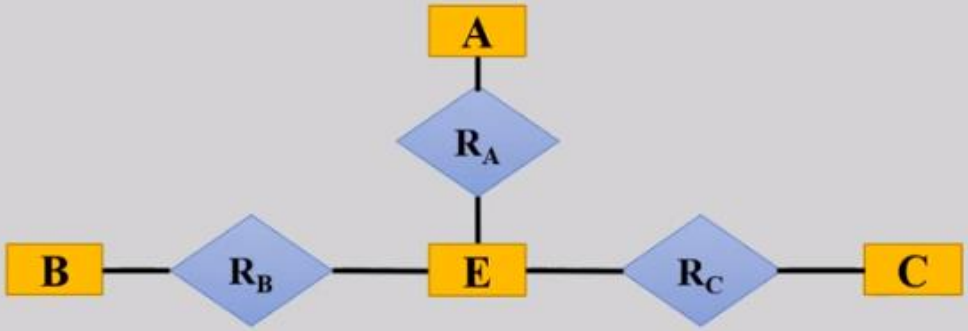
Generally, the relationships described in the databases are binary relationships. However, non-binary relationships can be represented by several binary relationships.

Ternary (n=3) relationship set R relating entity sets A, B and C.



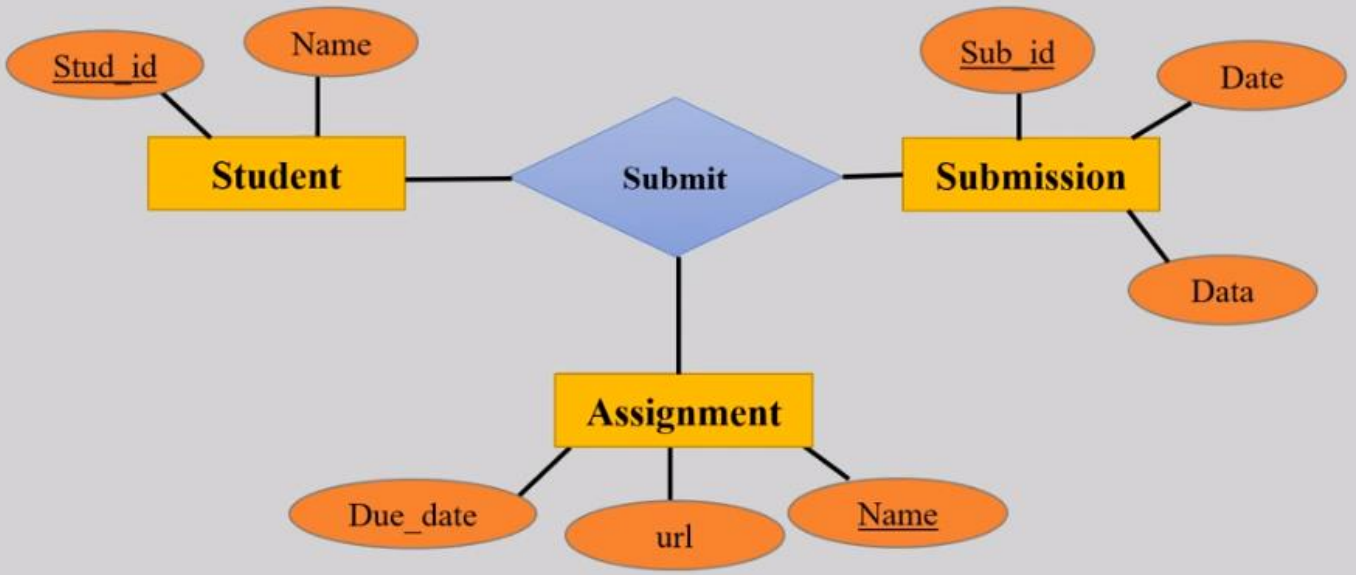
We replace the relationship set R by an entity set E,

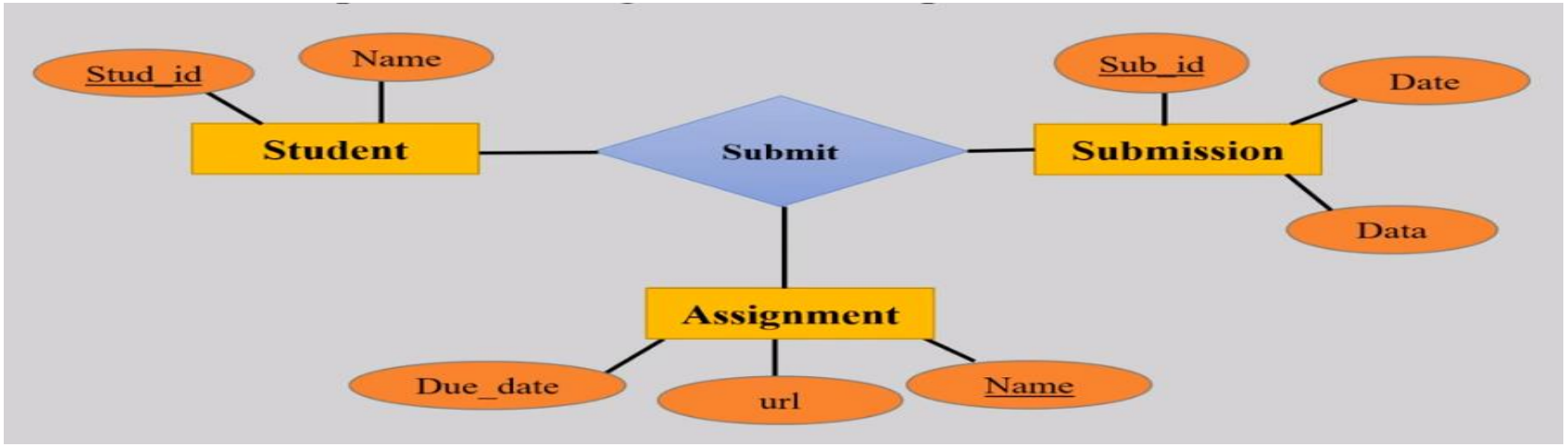
If the relationship set R had any attributes, these are assigned to entity set E.

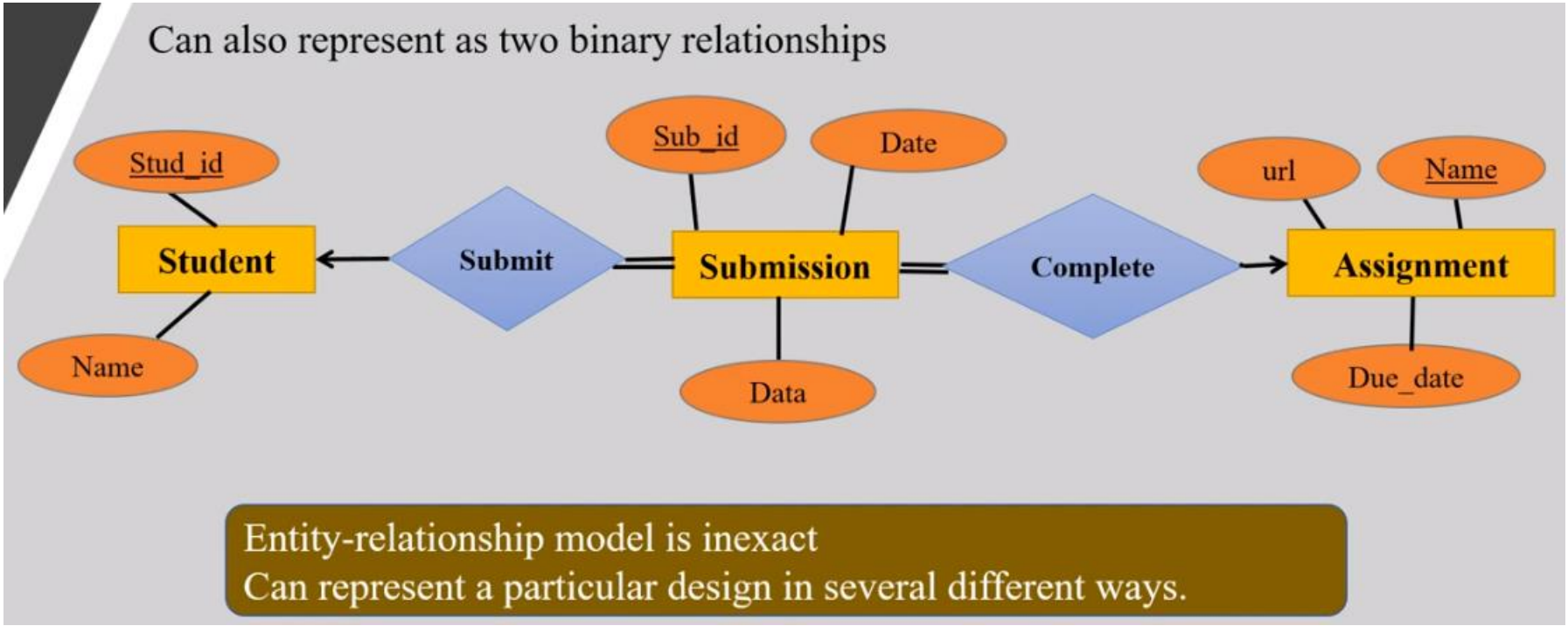


Example

Ternary relationship between student, assignment, and submission. Allow multiple submission for a particular assignment from a particular student.




Stud_id
Name
Student
Submit
Sub_id
Date
Submission
Data
Assignment
Due_date
url
Name

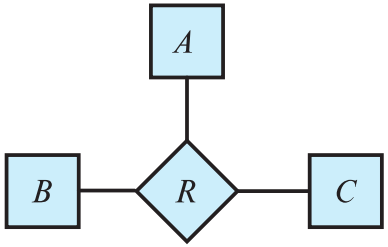

Can also represent as two binary relationships
Stud_id
Student
Name
Submit
Sub_id
Date
Submission
Data
Complete
url
Name
Assignment
Due_date
Entity-relationship model is inexact
Can represent a particular design in several different ways.

Converting Non-Binary Relationships to Binary Form

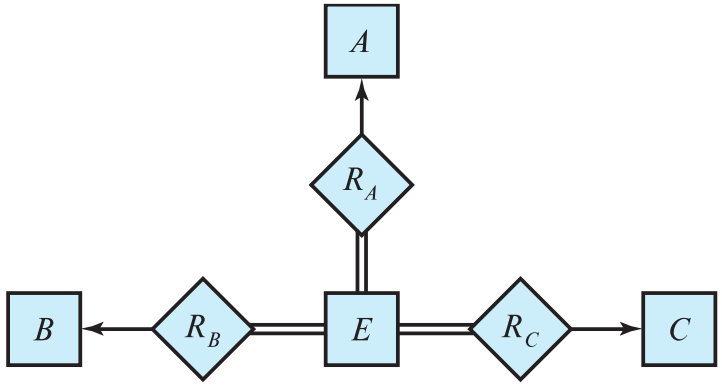
- In general, any non-binary relationship can be represented using binary relationships by creating an artificial entity set.
  - Replace R between entity sets A, B and C by an entity set E, and three relationship sets:

    1. R_A, relating E and A
    2. R_B, relating E and B
    3. R_C, relating E and C
  - Create an identifying attribute for E *and* add any attributes of R to E
  - For each relationship (a_i , b_i , c_i) in R, create

    1. a new entity e_i in the entity set E
    2. add (e_i , a_i) to R_A
    3. add (e_i , b_i) to R_B
    4. add (e_i , c_i) to R_C



(a)

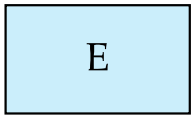


(b)

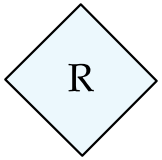
Converting Non-Binary Relationships (Cont.)

- Also need to translate constraints
  - Translating all constraints may not be possible
  - There may be instances in the translated schema that cannot correspond to any instance of *R*
    - Exercise: *add constraints to the relationships* R_A, R_B *and* R_C to ensure that a newly created entity corresponds to exactly one entity in each of entity sets *A, B* and *C*

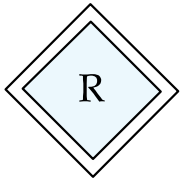
Summary of Symbols Used in E-R Notation



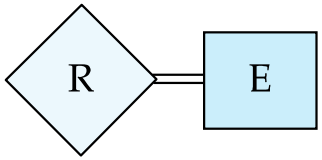
entity set



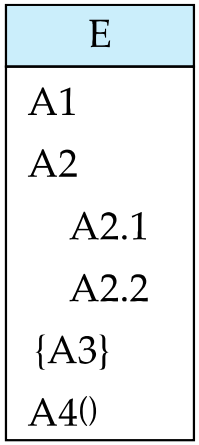
relationship set



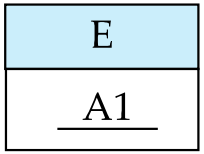
identifying relationship set for weak entity set



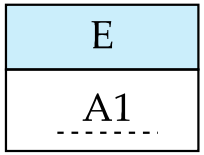
total participation of entity set in relationship



attributes:
simple (A1),
composite (A2) and
multivalued (A3)
derived (A4)

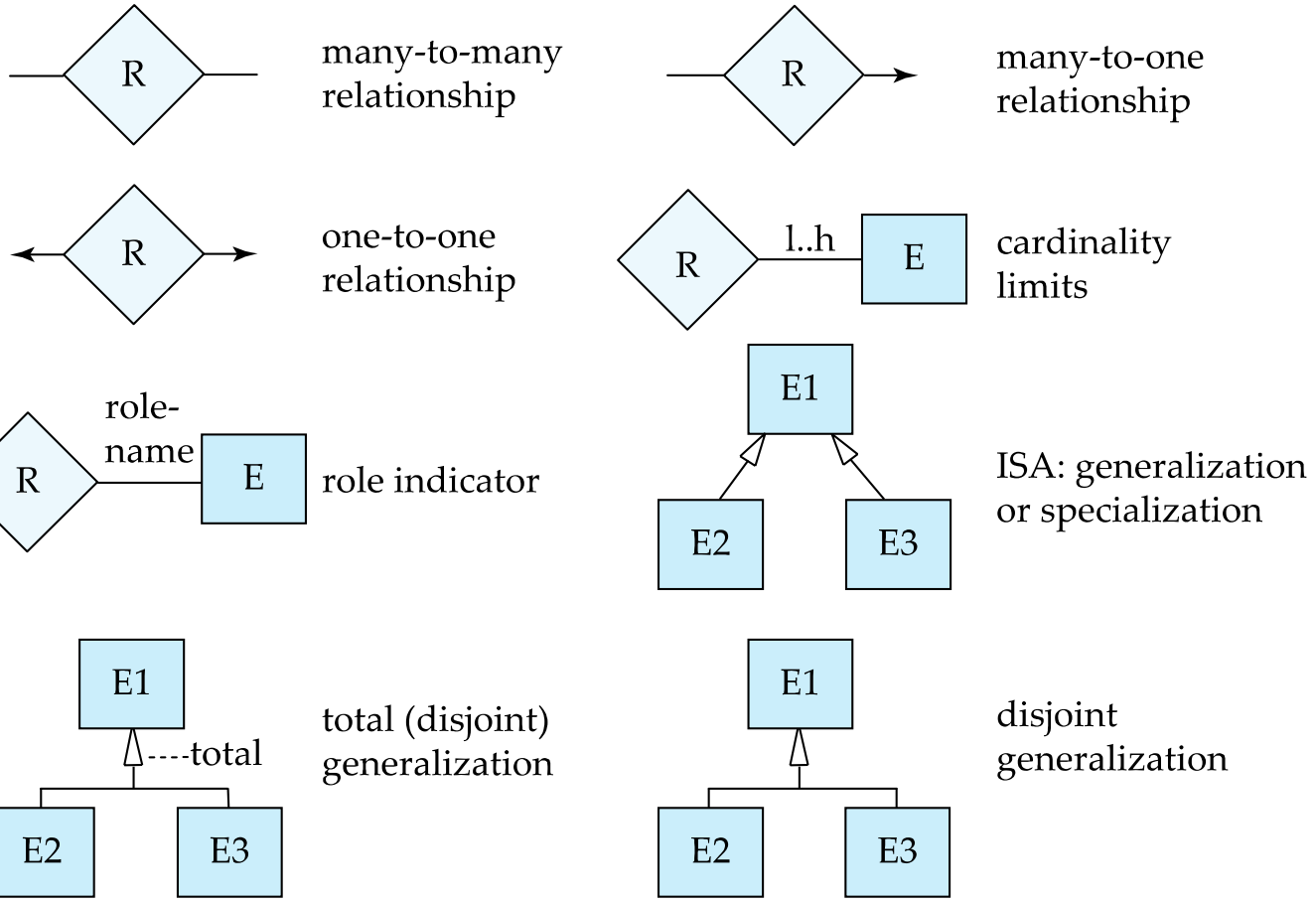


primary key



discriminating attribute of weak entity set

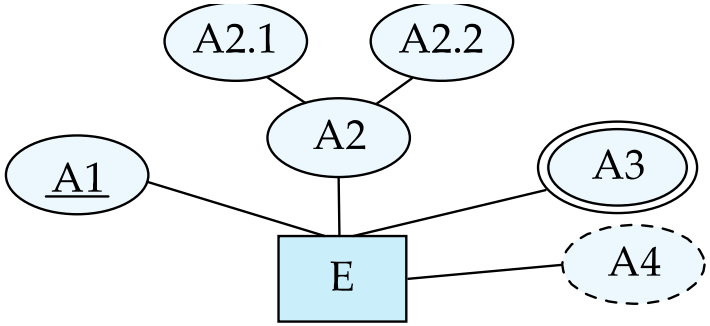
Symbols Used in E-R Notation (Cont.)



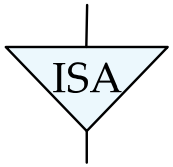
Alternative ER Notations

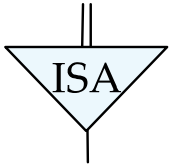
- Chen, IDE1FX, …

entity set E with
simple attribute A1,
composite attribute A2,
multivalued attribute A3,
derived attribute A4,
and primary key A1

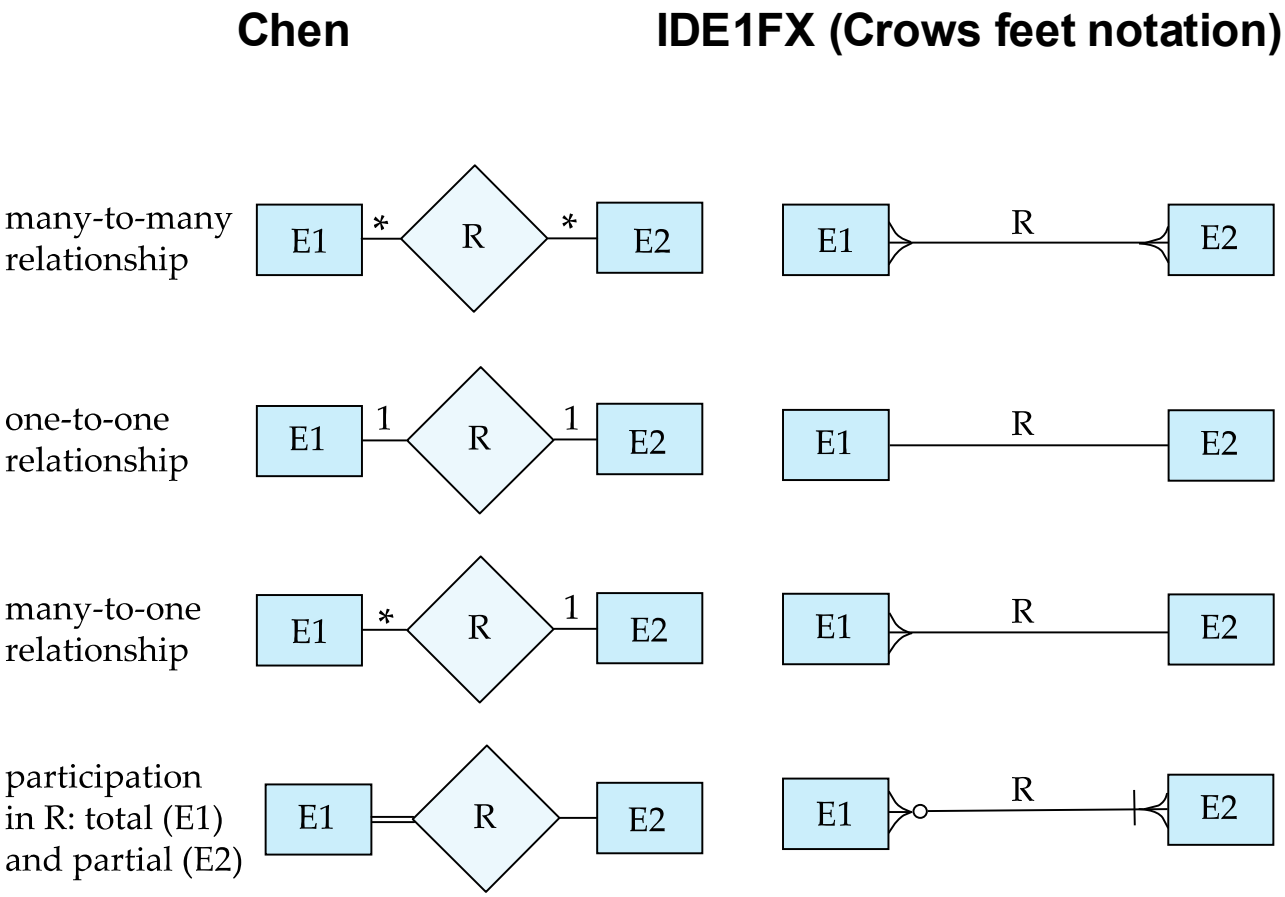


weak entity set 

generalization 

total generalization 

Alternative ER Notations

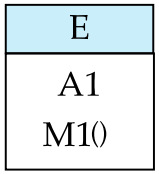


UML

- **UML**: Unified Modeling Language
- UML has many components to graphically model different aspects of an entire software system
- UML Class Diagrams correspond to E-R Diagram, but several differences.

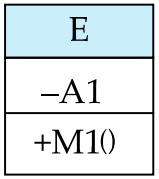
ER vs. UML Class Diagrams

ER Diagram Notation

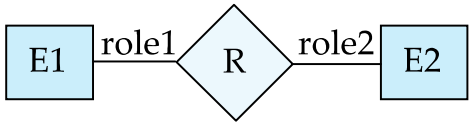


entity with attributes (simple, composite, multivalued, derived)

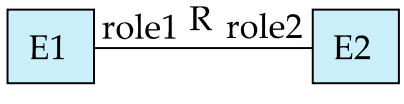
Equivalent in UML

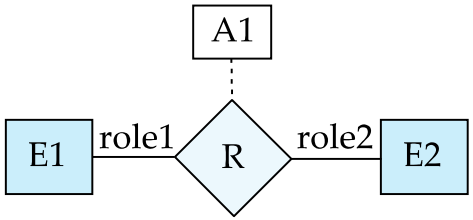


class with simple attributes and methods (attribute prefixes: + = public, – = private, # = protected)

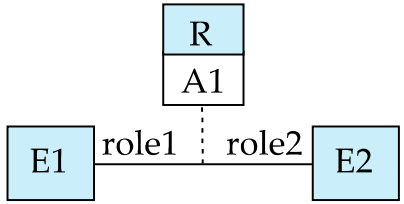


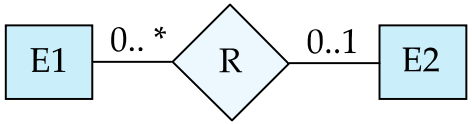
binary relationship



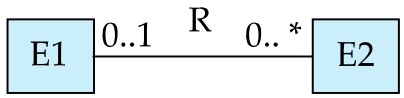


relationship attributes



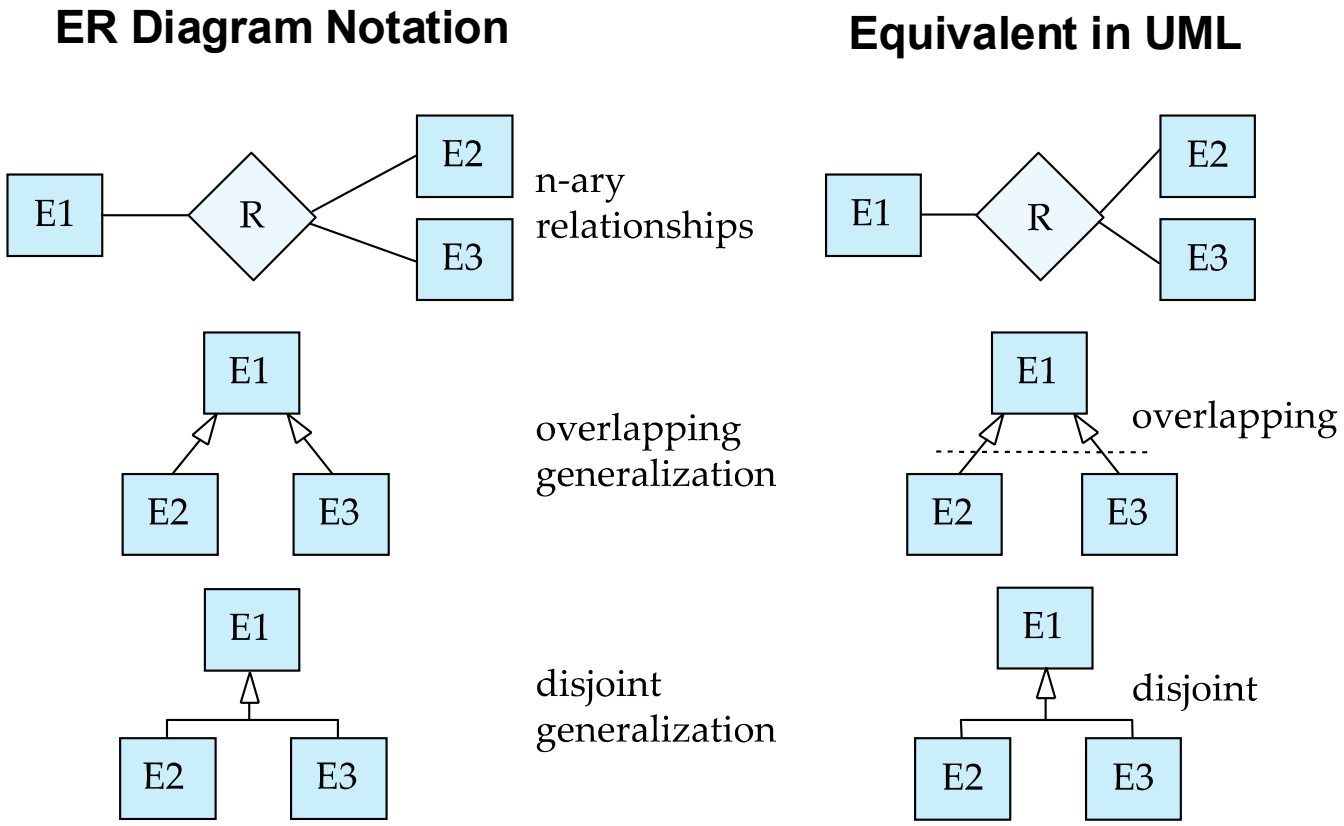


cardinality constraints



* Note reversal of position in cardinality constraint depiction

ER vs. UML Class Diagrams

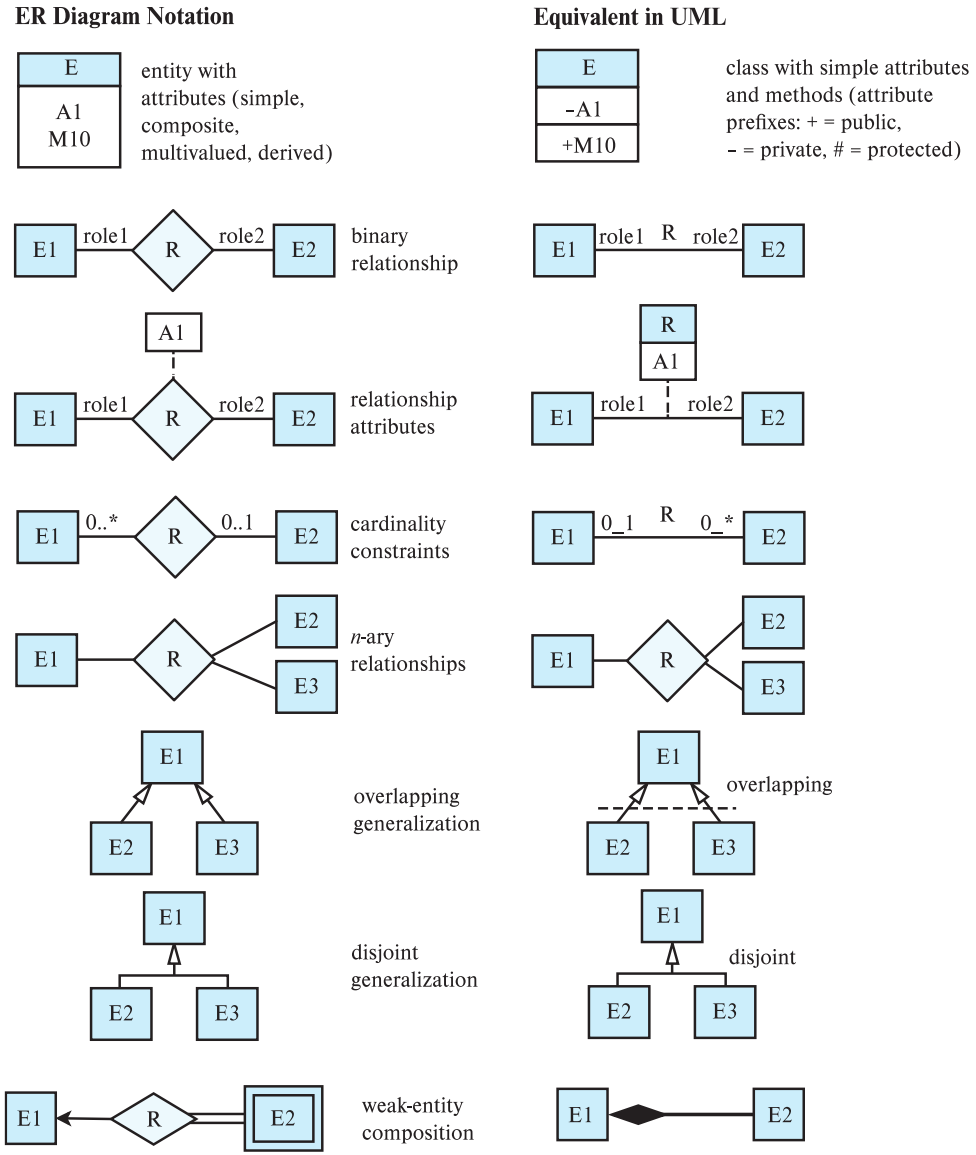


* Generalization can use merged or separate arrows independent of disjoint/overlapping

UML Class Diagrams (Cont.)

- Binary relationship sets are represented in UML by just drawing a line connecting the entity sets. The relationship set name is written adjacent to the line.
- The role played by an entity set in a relationship set may also be specified by writing the role name on the line, adjacent to the entity set.
- The relationship set name may alternatively be written in a box, along with attributes of the relationship set, and the box is connected, using a dotted line, to the line depicting the relationship set.

ER vs. UML Class Diagrams



End of Chapter 6