# TRANSFORM ER TO RELATIONAL SCHEMA

- This is the logical design step of the database design process.
- This step transforms ER Diagrams to relations.

# From ER Diagram to Relations (1)

| ER Concept | Relational Concept |
|---|---|
| Strong entity | Tuple |
| Weak entity | ??? |
| Strong entity set | Relation |
| Weak entity set | ??? |
| Attribute | Attribute |
| Key | Key |
| Composite attribute | ??? |
| Multi-valued attribute | ??? |

# From ER Diagram to Relations (2)

| ER Concept | Relational Concept |
|---|---|
| Unary 1-1 relationship | ??? |
| Unary 1-m relationship | ??? |
| Unary m-m relationship | ??? |
| Binary 1-1 relationship | ??? |
| Binary 1-m relationship | ??? |
| Binary m-m relationship | ??? |
| Ternary relationship | ??? |
| IS_A hierarchy | ??? |

# From ER Diagram to Relations (3)

Summary:

- ER model is more semantically oriented than the relational model.
- Some ER concepts, e.g., connectivity constraints, cannot be expressed in the relational model.
- Are there important concepts that are supported by the relational model but not by the ER model?
  - Foreign key

# Basic Ideas of the Transformation

Entity ====> Tuple

Entity set ====> Relation

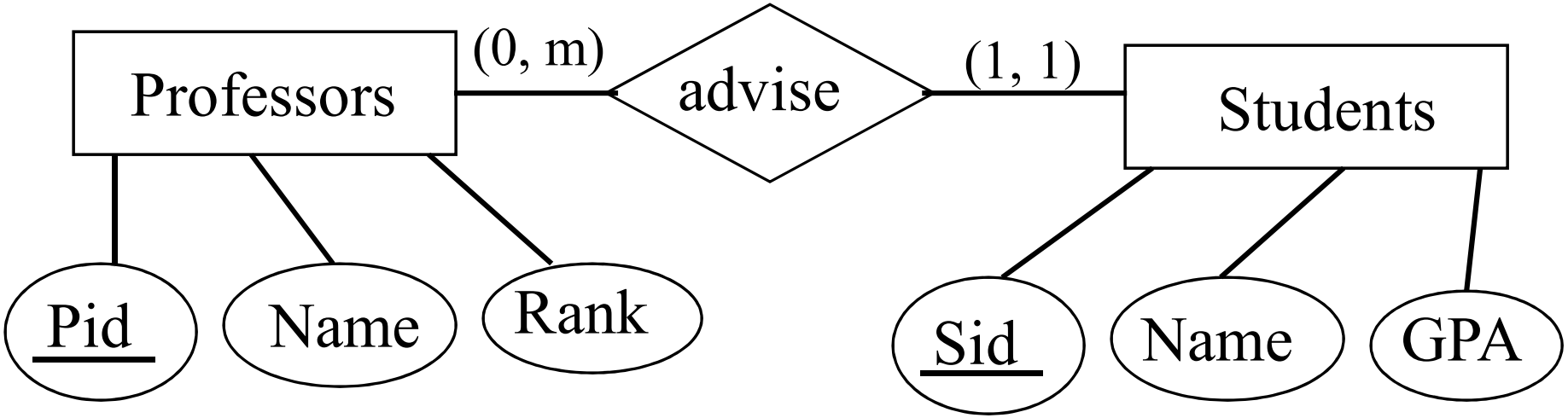
Attribute ====> Attribute

Key ====> Key

Relationship ====> Tuple or foreign key value(s)

Relationship set ====> Relation or foreign key(s)

# An Example (1)




| Professors | Advise | Students |
|---|---|---|
| p1: 123, Jack, Prof. | p1 advises s1 | s1: 456, John, 3.4 |
| p2: 234, Ann, Prof. | p1 advises s2 | s2: 567, Carl, 3.2 |
| p3: 345, Bob, Prof. | p3 advises s3 | s3: 678, Ken, 3.5 |

# An Example (2)

Transform the ER diagram into three relations:

**Professors**

| Pid | Name | Rank |
|---|---|---|
| 123 | Jack | Prof. |
| 234 | Ann | Prof. |
| 345 | Bob | Prof. |

**Advise**

| Pid | Sid |
|---|---|
| 123 | 456 |
| 123 | 567 |
| 345 | 678 |

**Students**

| Sid | Name | GPA |
|---|---|---|
| 456 | John | 3.4 |
| 567 | Carl | 3.2 |
| 678 | Ken | 3.5 |

# An Example (3)

Two relations are sufficient:

**Professors**

| Pid | Name | Rank |
|---|---|---|
| 123 | Jack | Prof. |
| 234 | Ann | Prof. |
| 345 | Bob | Prof. |

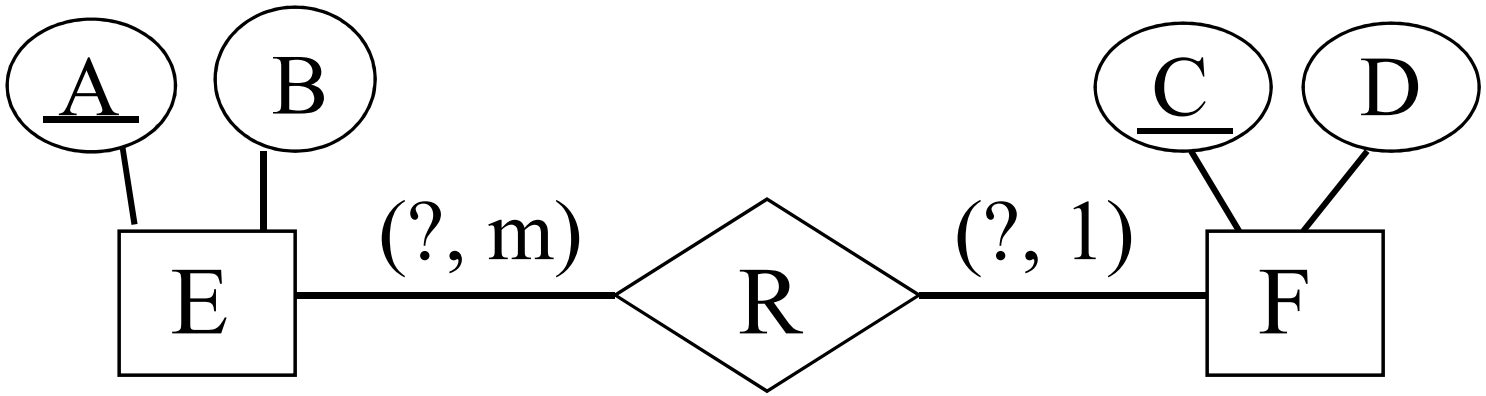
**Students**

| Sid | Name | GPA | Pid |
|---|---|---|---|
| 456 | John | 3.4 | 123 |
| 567 | Carl | 3.2 | 123 |
| 678 | Ken | 3.5 | 345 |

# From ER Diagram to Relations

*Transformation Guideline 1*: Transform relationship sets to foreign key attributes whenever possible.

# Transform Binary Relationship (1)

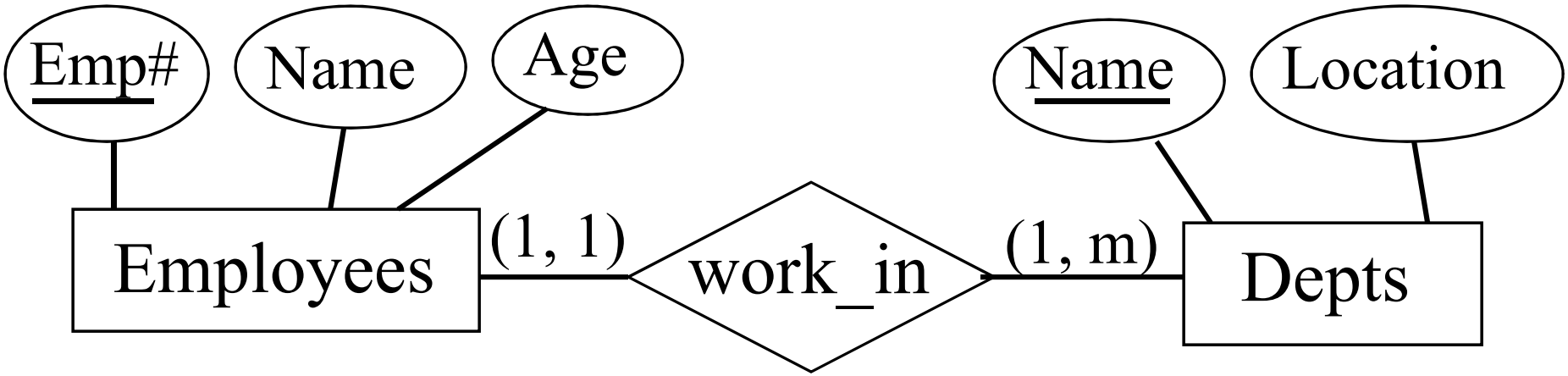



Case 1: one-to-many relationship

===> E(A, B), F(C, D, A)

- Relationship R is transformed to a foreign key.
- The foreign key goes to the side with max_card = 1

# Transform Binary Relationship (2)

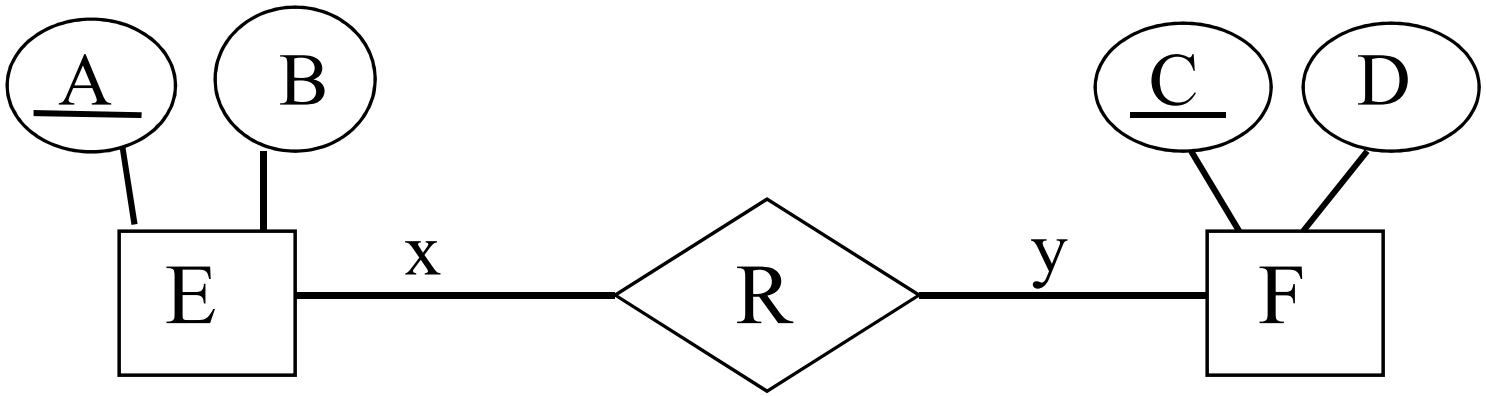



==> Depts(Name, Location)

Employees(Emp#, Name, Age, Dept_name)

*Renaming* is useful for improving understandability.

# Transform Binary Relationship (3)



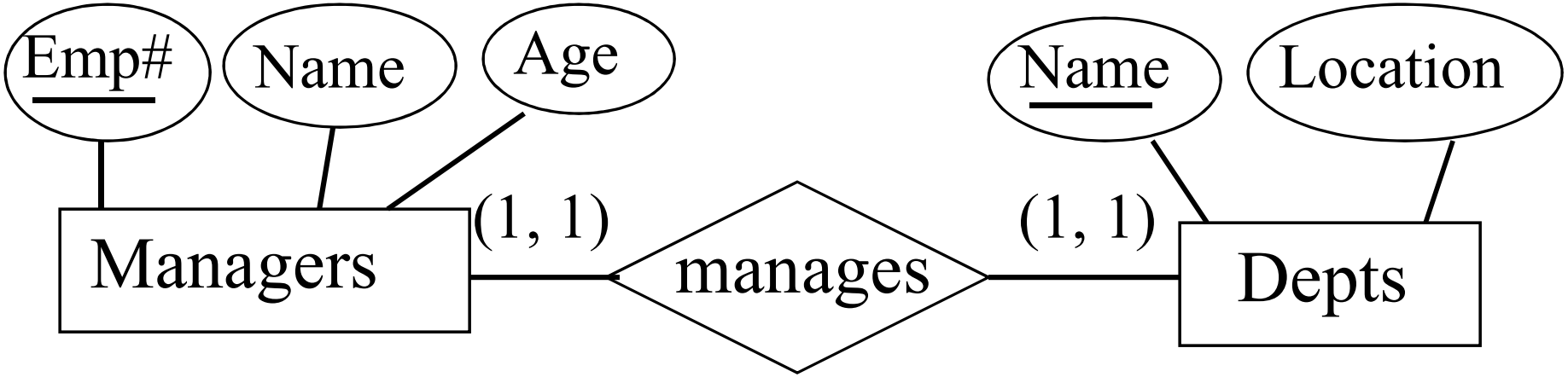

Case 2: one-to-one relationship

Case 2.1: x = (1, 1) and y = (1, 1)

===> E(A, B), F(C, D, A) or

===> E(A, B, C), F(C, D)

# Transform Binary Relationship (4)



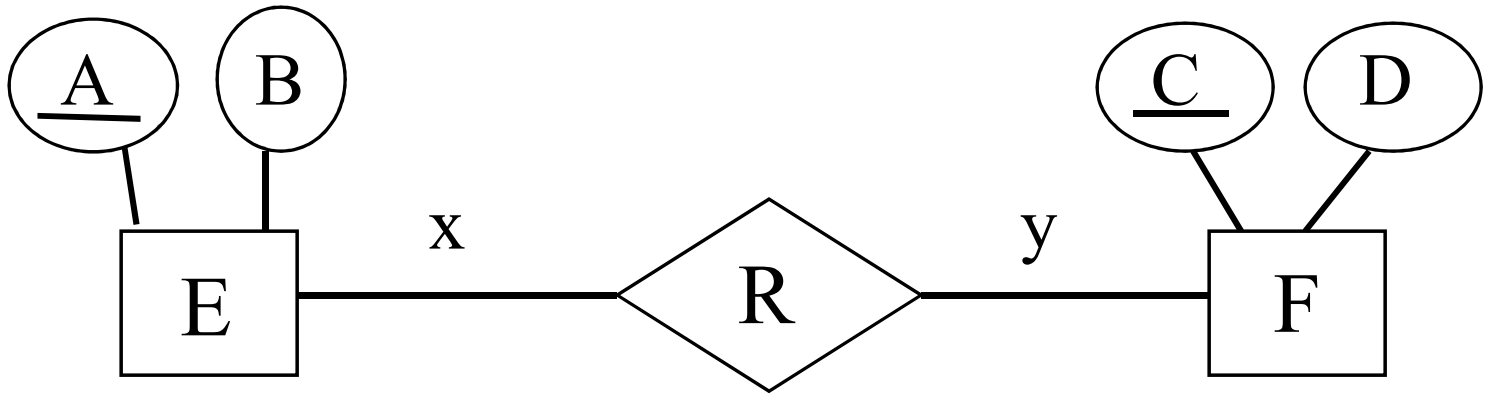

==> Depts(Name, Location)

Managers(Emp#, Name, Age, Dept_name)

==> Depts(Name, Location, Manager_Emp#)

Managers(Emp#, Name, Age)

# Transform Binary Relationship (5)

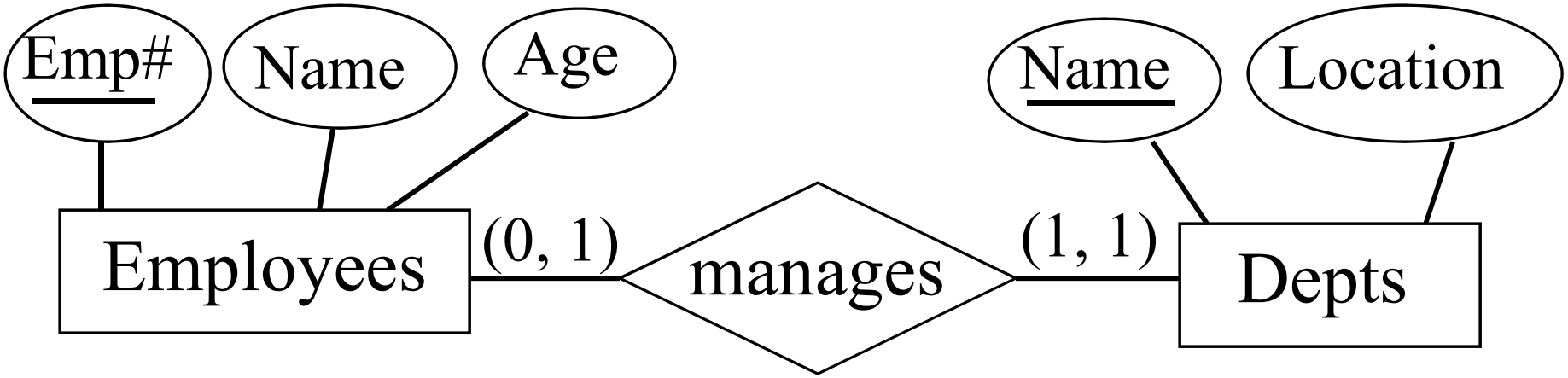



Case 2.2: x = (0, 1) and y = (1, 1)

===> E(A, B), F(C, D, A)

- The entity set with the total participation is transformed to a relation with a foreign key.

# Transform Binary Relationship (6)




==> Depts(Name, Location, Manager_Emp#)

Employees(Emp#, Name, Age)

- Why not let Employees have the foreign key?

# Transform Binary Relationship (7)

*Transformation Guideline 2*: Avoid introducing null values as much as possible. When it is necessary to introduce null values, introduce as few null values as possible.

# Transform Binary Relationship (5)

Case 2.3: x = (0, 1) and y = (0, 1)

- Transform the relationship set into a foreign key attribute.
- The foreign key should go to the relation that causes the least number of null values.

# Transform Binary Relationship (9)

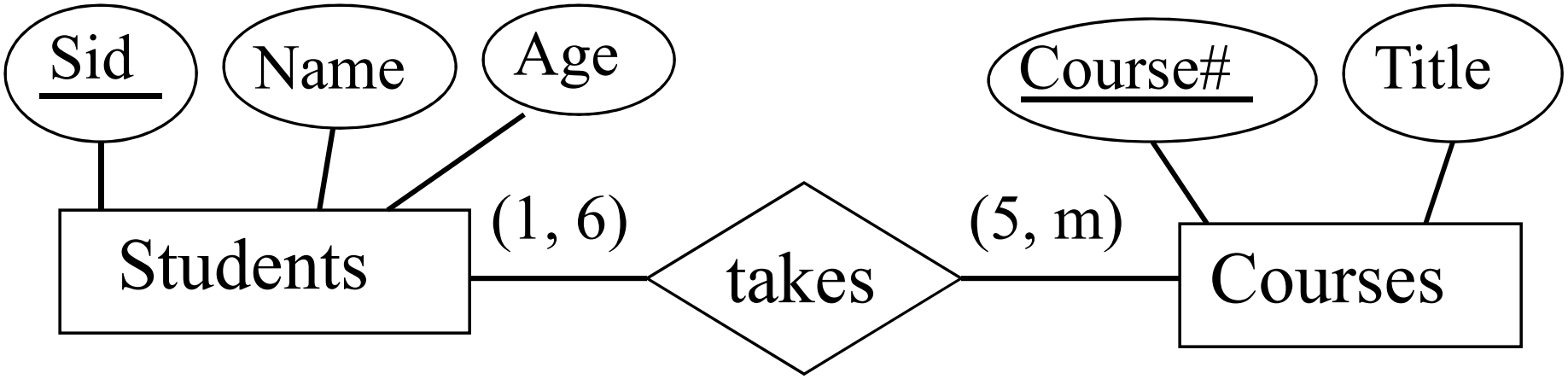
Case 3: many-to-many relationship

Case 3.1: R has no attribute

===> E(A, B), F(C, D), R(A, C)

- Transform the m-to-m relationship to a separate relation.
- R has two foreign keys.
- The key of R consists of the foreign keys.

# Transform Binary Relationship (10)




==> Students(Sid, Name, Age)

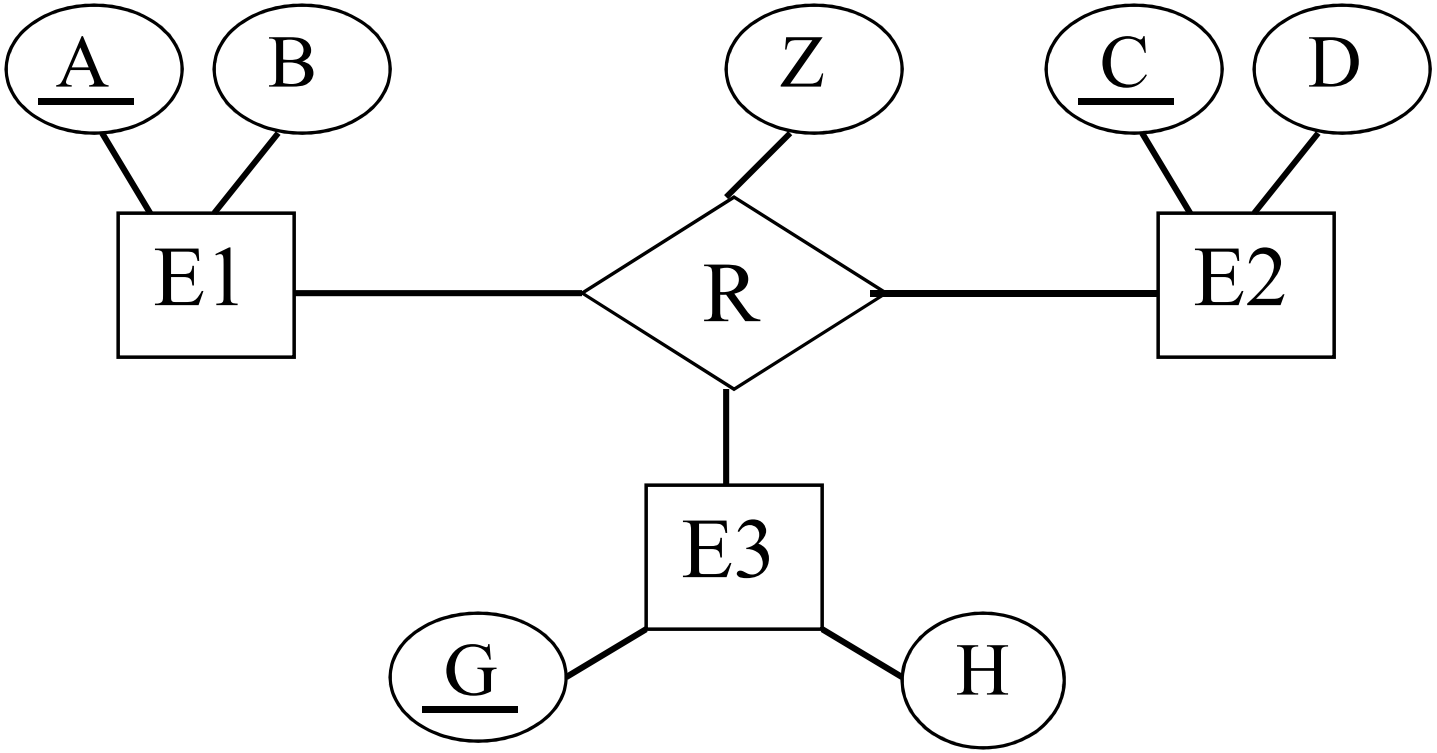
Courses(Course#, Title)

Takes(Sid, Course#)

Case 3.2: R has attribute Z

===> E(A, B), F(C, D), R(A, C, Z)

# Transform Ternary Relationship

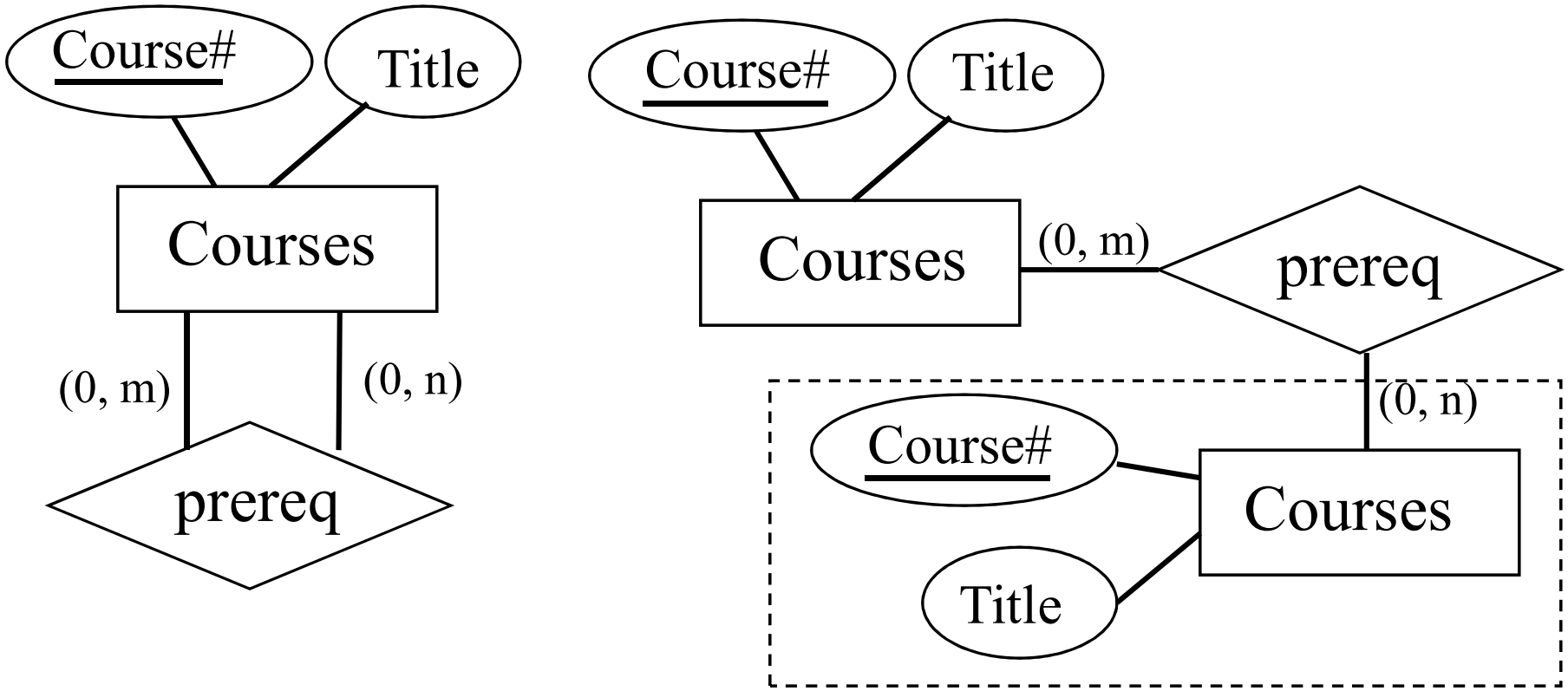



==> E1(A, B), E2(C, D), E3(G, H),
R(A, C, G, Z)

# Transform Unary Relationship (1)

- Create a shadow entity set and transform the unary relationship into a binary relationship.
- Apply the rules for transforming binary relationships.
- After the transformation, remove one redundant relation, or if there is no redundant relation, remove the relation with fewer attributes.

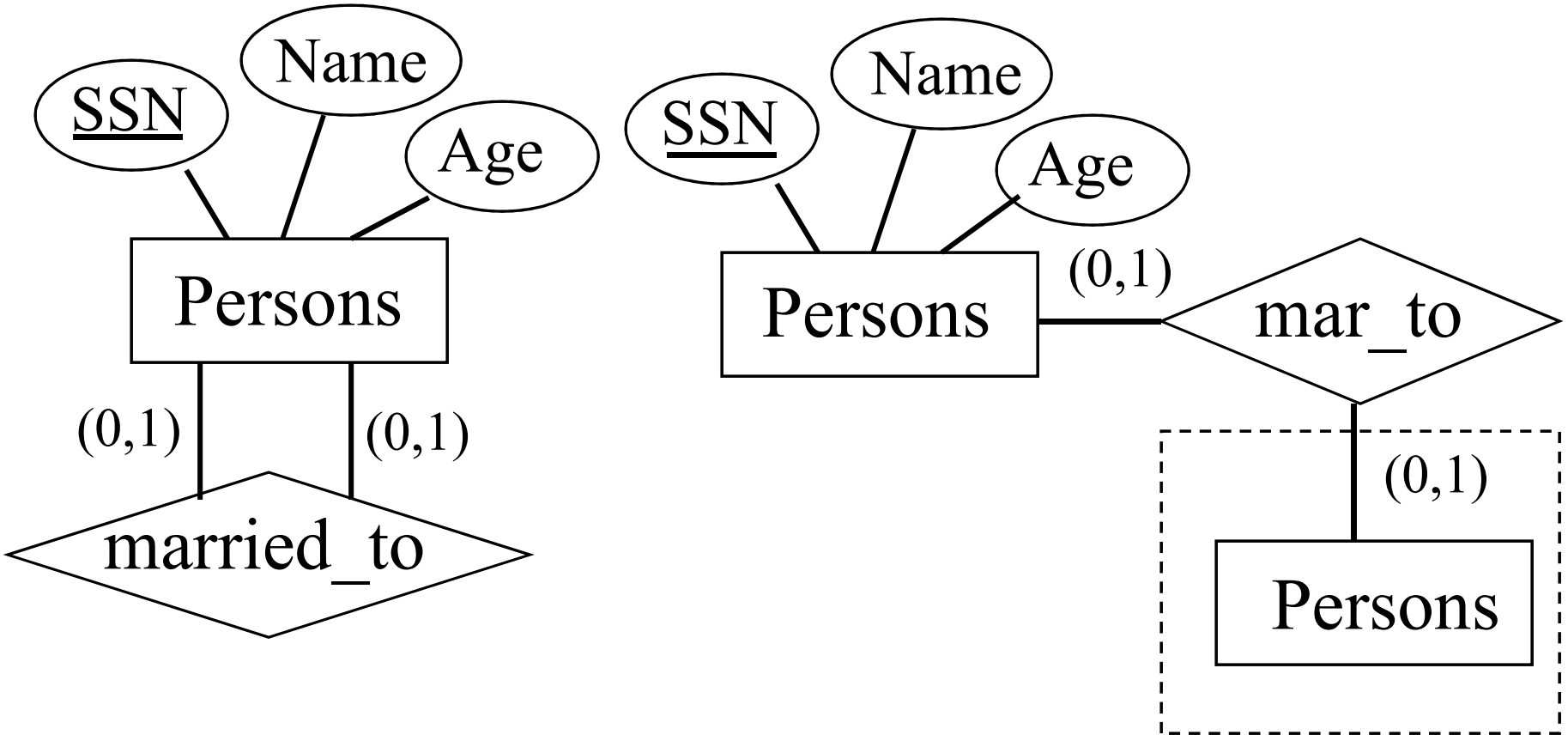
# Transform Unary Relationship (2)




Courses(Course#, Title)

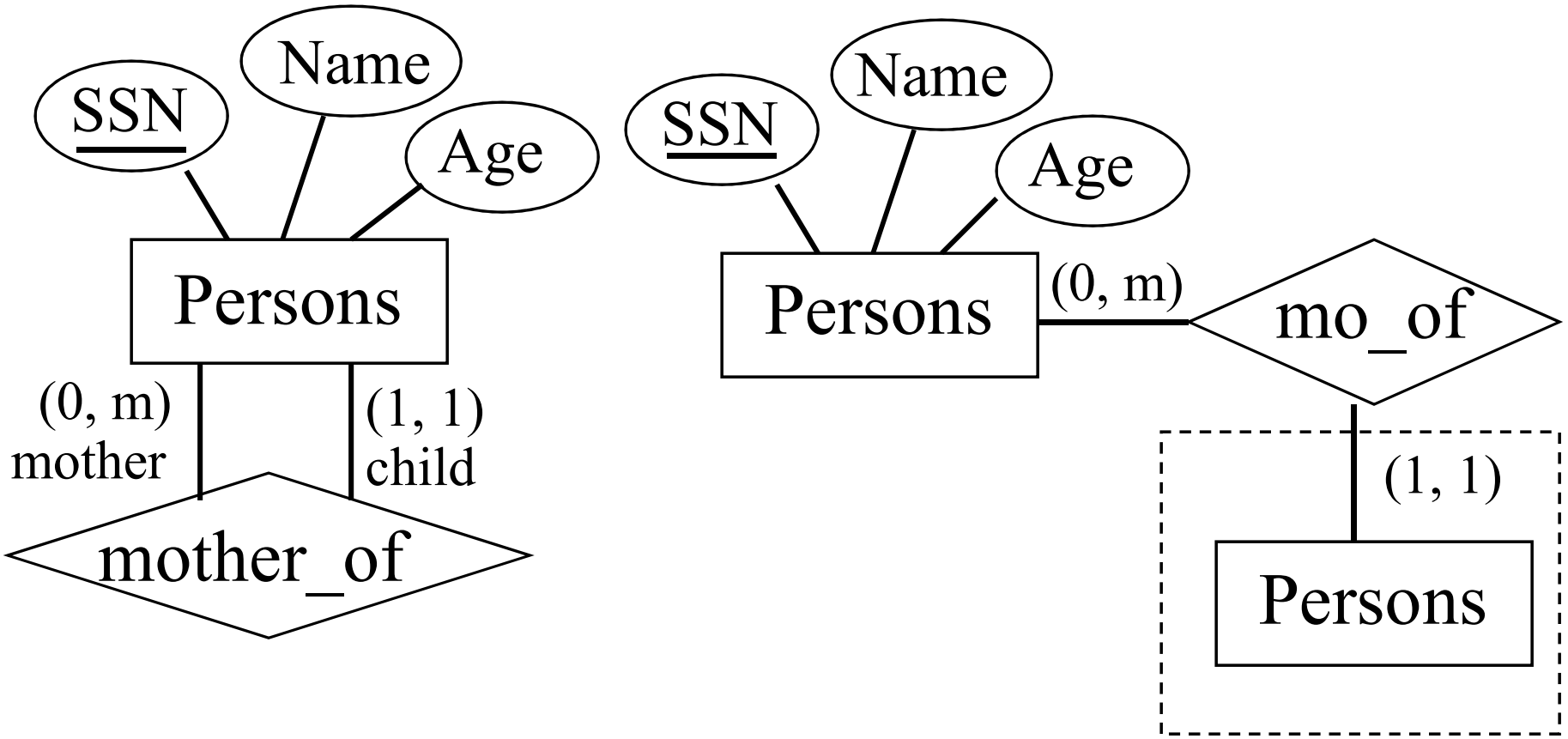
Prereq(Course#, Prereq_Course#)

# Transform Unary Relationship (3)




Persons(SSN, Name, Age, Spouse_SSN)

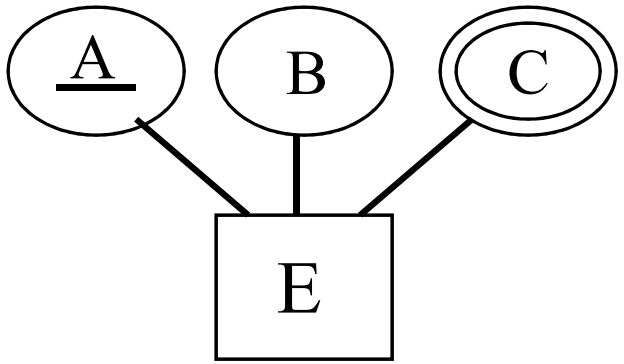
# Transform Unary Relationship (4)




Persons(SSN, Name, Age, Mother_SSN)

# Transform Multi-valued Attribute (1)

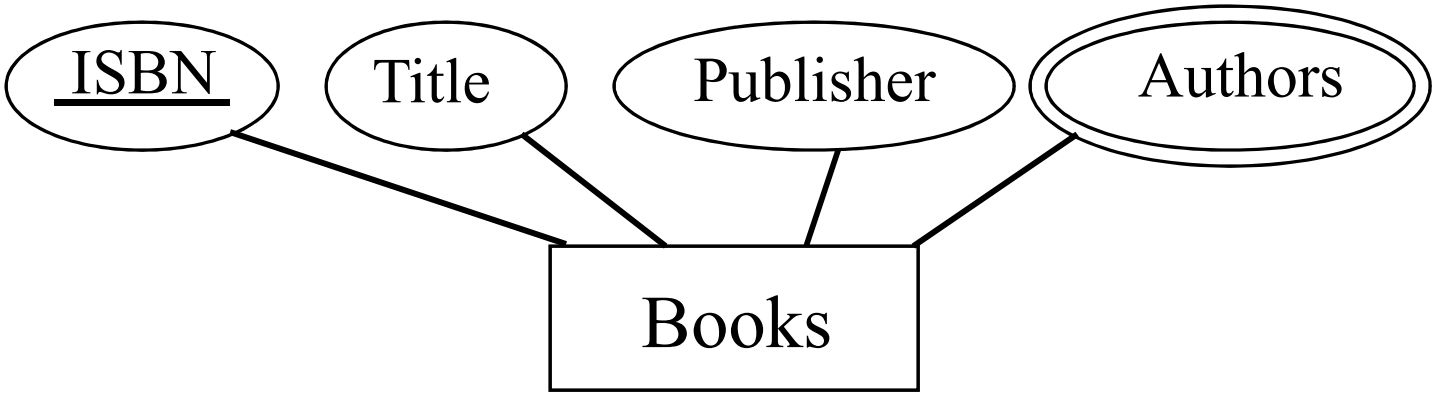
- Create a separate relation for each multi-valued attribute.




E(A, B), E_C(A, C)

- E_C.A should be defined to be a foreign key referencing E.A

# Transform Multi-valued Attribute (2)



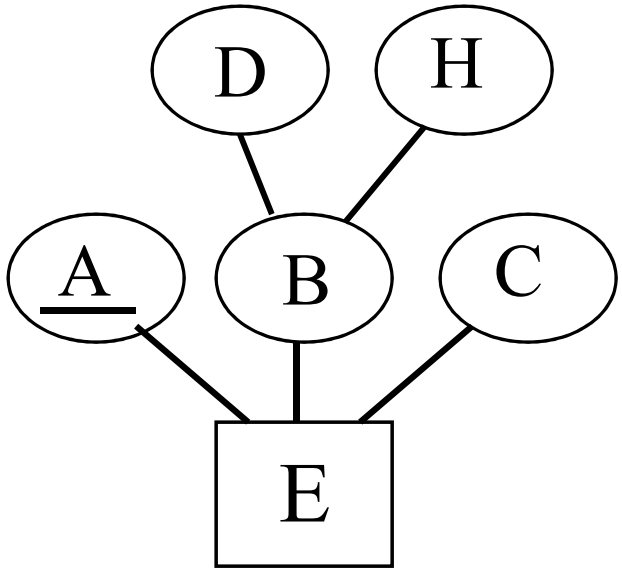

==> Books (ISBN, Title, Publisher)

Book_Authors (ISBN, Author)

- Define Book_Authors.ISBN as a foreign key referencing Books.ISBN

# Transform Composite Attribute (1)

Method 1: Use only simple attributes and ignore the composite attribute.

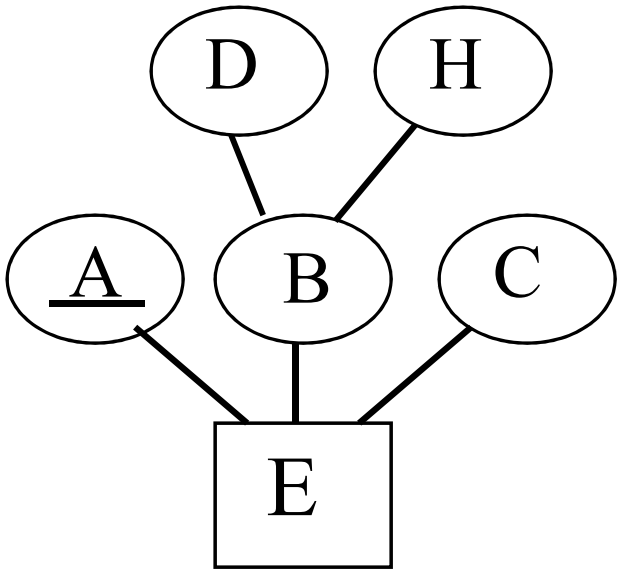



==>

E(A, D, H, C)

# Transform Composite Attribute (2)

Method 2: Transform the composite attribute to a separate relation.

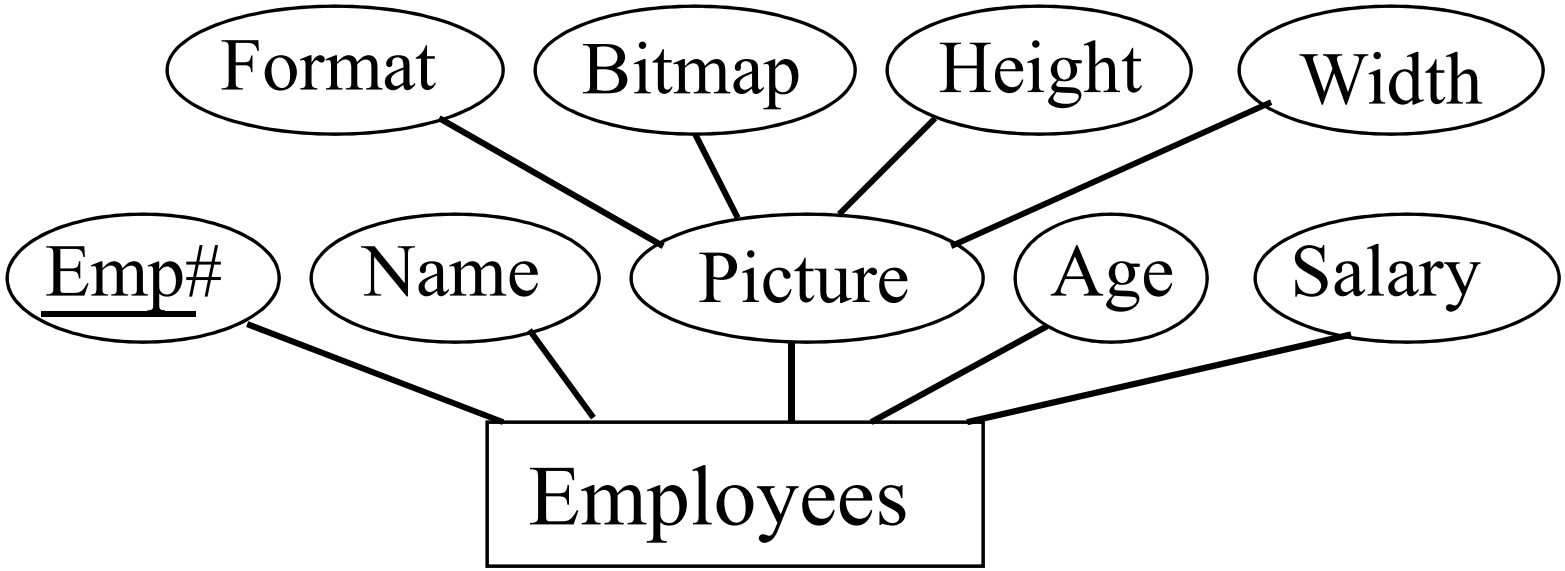



==>

E(A, C),
E_B (A, D, H)

# Transform Composite Attribute (3)

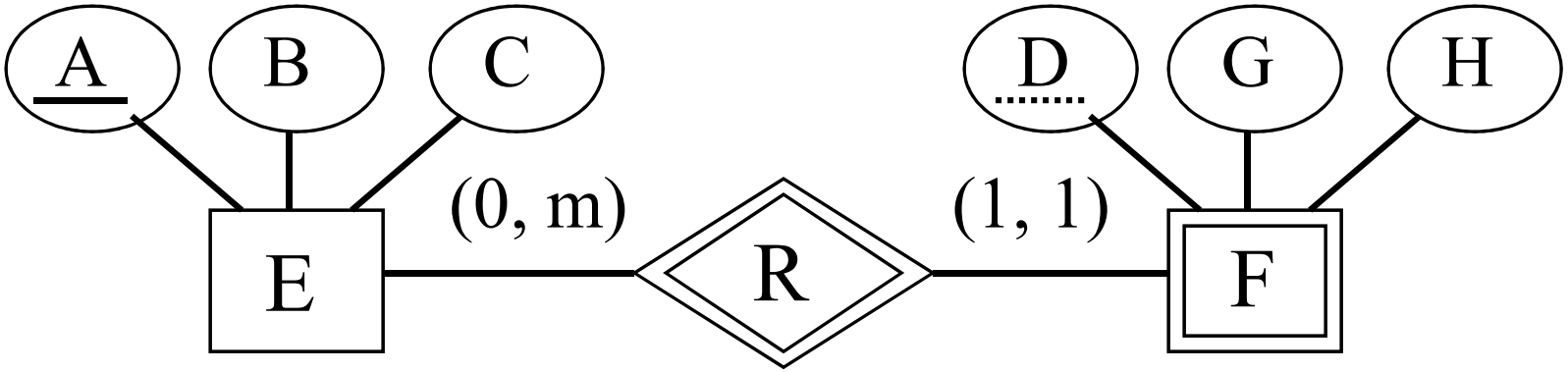
An Example using method 2:




Employees (Emp#, Name, Age, Salary)

Emp_Pic (Emp#, Bitmap, Format, Height, Width)

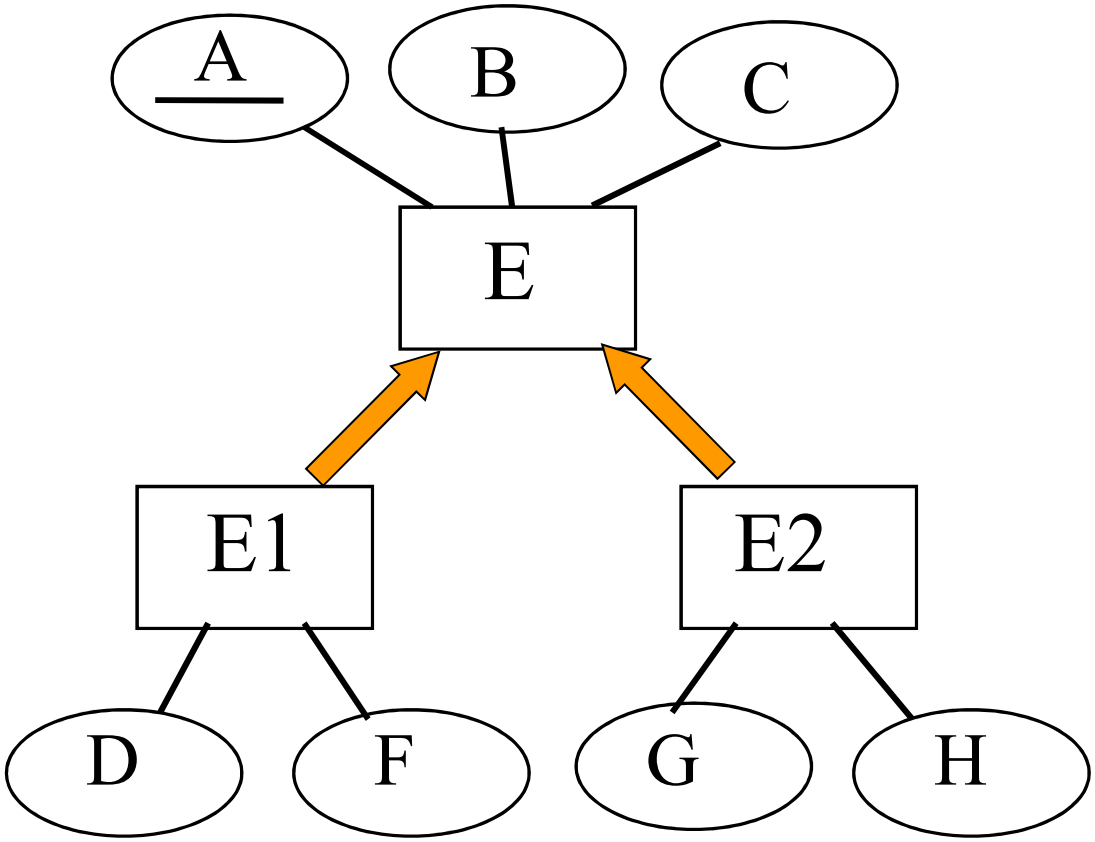
# Transform Weak Entity Set




==> E (A, B, C), F(A, D, G, H)

- The key of F consists of the key of E and the partial key of F.
- F.A is a foreign key referencing E.A

# Transform IS_A Hierarchy (1)



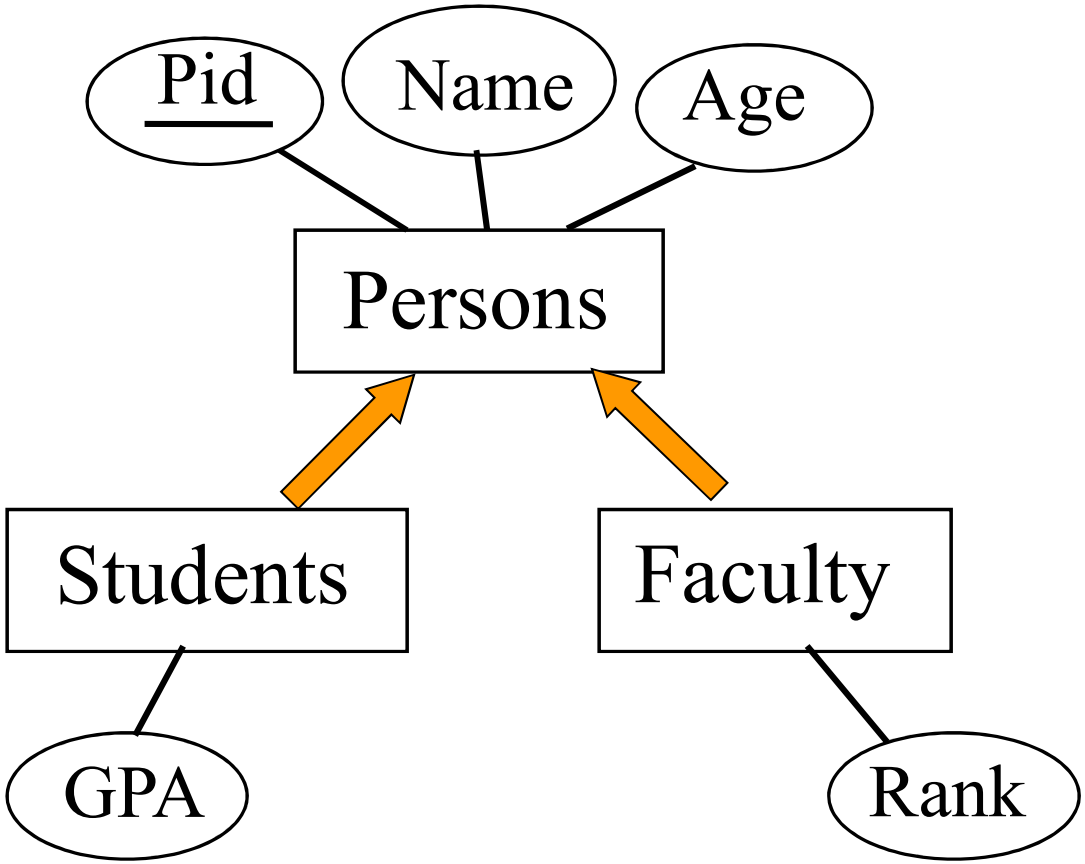
# Transform IS_A Hierarchy (2)

*Method 1*: For general case

==> E(A, B, C), E1(A, D, F),
E2(A, G, H)

- Only the key is explicitly inherited from the super entity set.
- A tuple in E either corresponds to an entity in E or an entity in a sub entity set.
- E1.A and E2.A are defined to be foreign keys referencing E.A.

# Transform IS_A Hierarchy (5)



# Transform IS_A Hierarchy (6)

Real world information:

| | Pid | Name | Age | GPA | Rank |
|---|---|---|---|---|---|
| stud: | 123456789 | John | 27 | 3.5 | |
| facul: | 234567891 | Bill | 43 | | Prof. |
| staff: | 345678912 | Mary | 37 | | |

# Transform IS_A Hierarchy (7)

*Method 1:*

**Persons**

| Pid | Name | Age |
|---|---|---|
| 123456789 | John | 27 |
| 234567891 | Bill | 43 |
| 345678912 | Mary | 37 |

**Students**

| Pid | GPA |
|---|---|
| 123456789 | 3.5 |

**Faculty**

| Pid | Rank |
|---|---|
| 234567891 | Prof. |

# Transform IS_A Hierarchy (3)

*Method 2*: When the union of the sub entity sets contains the same set of entities as the super entity set.

==> E1(A, D, F, B, C), E2(A, G, H, B, C)

- All attributes are explicitly inherited from the super entity set.
- In principle, this method could also be used when the union of the sub entity sets is not a superset of the super entity set. In this case:

==> E(A, B, C), E1(A, D, F, B, C),
E2(A, G, H, B, C)

# Transform IS_A Hierarchy (8)

*Method 2:*

**Persons**

| Pid | Name | Age |
|---|---|---|
| 345678912 | Mary | 37 |

**Students**

| Pid | Name | Age | GPA |
|---|---|---|---|
| 123456789 | John | 27 | 3.5 |

**Faculty**

| Pid | Name | Age | Rank |
|---|---|---|---|
| 234567891 | Bill | 43 | Prof. |

# Transform IS_A Hierarchy (4)

*Method 3*: When the sub entity sets are disjoint based on the values of an (implicit) attribute K.

==> E(A, B, C, D, F, G, H, K)

- K has the same value for entities from the same entity set but different values for entities from different entity sets.

E.g.: Super entity set: Employees

Sub entity sets: Programmers, Analysts, …

K: Position

- For entities from a sub entity set, fill missing attributed with null values.

# Transform IS_A Hierarchy (9)

*Method 3:*

**Persons**

| Pid | Name | Age | GPA | Rank | Type |
|---|---|---|---|---|---|
| 123456789 | John | 27 | 3.5 | | Student |
| 234567891 | Bill | 43 | | Prof. | Faculty |
| 345678912 | Mary | 37 | | | Staff |

# Transform IS_A Hierarchy (10)

Comparison of the three methods

- Method 3 results in the smallest number of tables, followed usually by method 2 and then method 1.
  - Why?
- Method 3 may introduce significant amount of null values while methods 1&2 don't.
  - Why?
- Method 2 usually leads to smaller tables than methods 1&3.
  - Why?

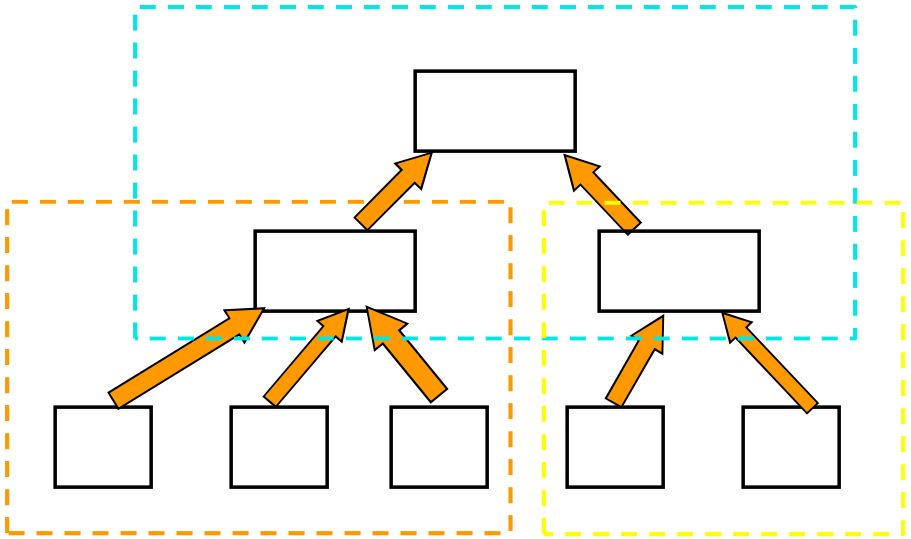
# Transform IS_A Hierarchy (11)

Factors that may influence the choice of method

- The number of m-to-m & ternary relationships and multi-valued attributes of super entity set.
  - Large number favors methods 1&3. Why?
- The number of attributes and relationships of the sub entity sets.
  - Large number favors methods 1&2. Why?
- The number of features shared by sub entity sets.
  - Small number favors method 2. Why?
- Whether or not every entity in the super entity set is contained in one of the sub entity sets.
  - Yes favors method 2. Why?

# Transform IS_A Hierarchy (12)

- If an IS-A hierarchy has more than two levels, it is possible to apply a different basic transformation method to transform a different portion of the hierarchy.



# Transform a Complex ER Diagram (1)

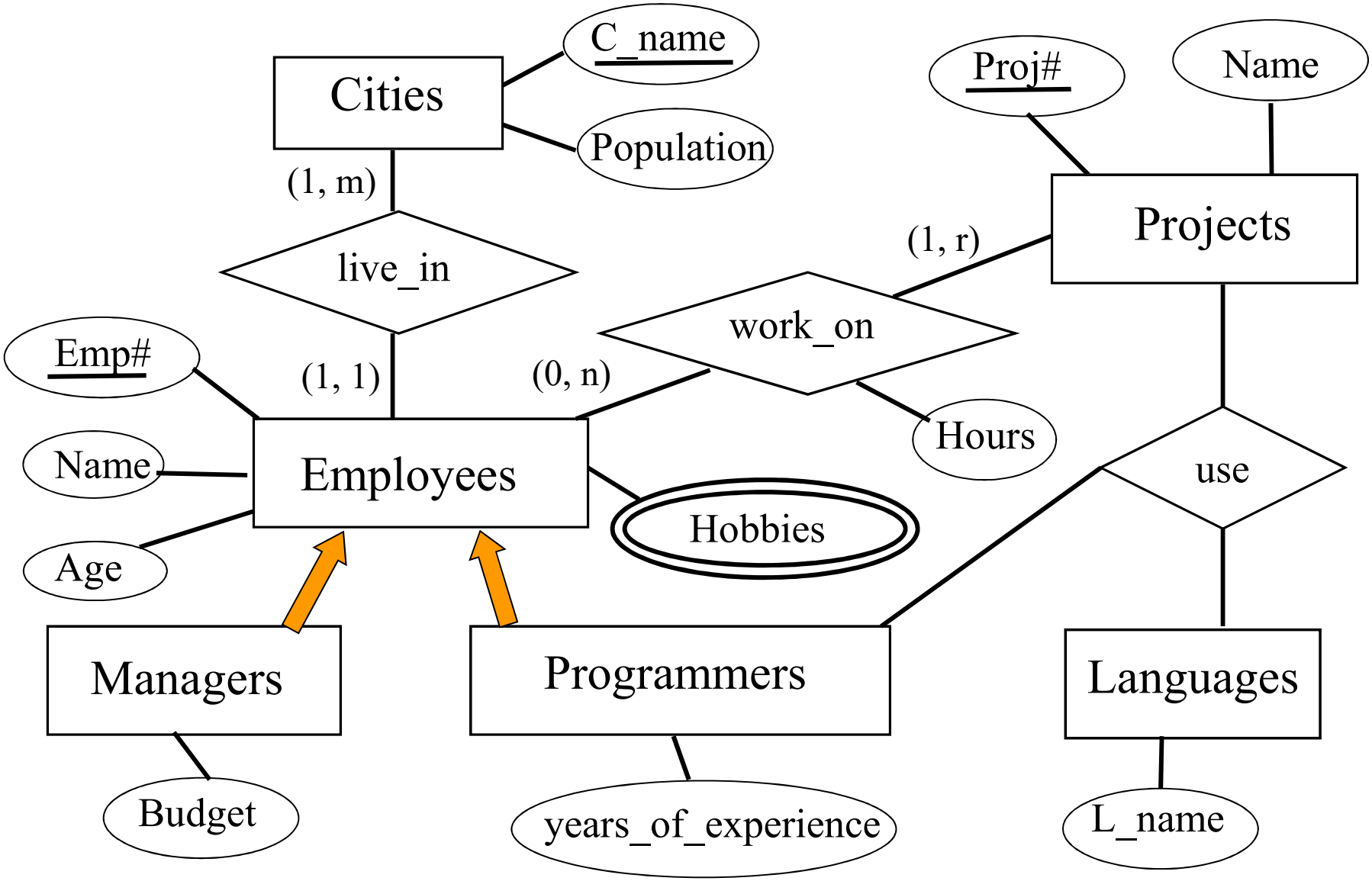
A general procedure:

- Transform each entity set into a relation (excluding multi-valued and composite attributes).
  - Transform each IS_A hierarchy
    - **Consider two adjacent levels (parent-child) at a time.**
    - **For methods 1 and 2, transform entity sets in a top-down manner (keep inheritance in mind).**
    - **For method 3, transform entity sets in a bottom-up matter (keep reverse inheritance in mind).**
  - Transform each multi-valued attribute into a separate relation.
  - Transform each composite attribute (select appropriate method).
  - Specify the key for each relation.

# Transform a Complex ER Diagram (2)

- Transform each relationship set.
  - For any unary/binary 1-to-1 or 1-to-m relationship, transform it by adding a foreign key to an appropriate relation.
  - Transform any m-to-m or high degree (degree > 2) relationship by creating a separate relation. Specify the key.
  - Re-visit relations involved in IS_A hierarchies to deal with feature inheritance.
  - Specify foreign keys.

# Transform a Complex ER Diagram (3)



# Transform a Complex ER Diagram (4)

Use method 1:

Employees(Emp#, Name, Age, C_name)

Employee-Hobby(Emp#, Hobby)

Managers(Emp#, Budget)

Programmers(Emp#, Years_of_experience)

Cities(C_name, Population)

Projects(Proj#, Name)

Languages(L_name)

Work_on(Emp#, Proj#, Hours)

Use(Emp#, Proj#, L_name)

# Transform a Complex ER Diagram (5)

Use method 1 (continued):

- Employee-Hobby.Emp#, Managers.Emp#, Programmers.Emp# and Work_on.Emp# are foreign keys referencing Employees.Emp#
- Use.Emp# is a foreign key referencing Programmers.Emp#
- Employees.C_name is a foreign key referencing Cities.C_name
- Work_on.Proj# and Use.Proj# are foreign keys referencing Projects.Proj#
- Use.L_name is a foreign key referencing Languages.L_name

# Transform a Complex ER Diagram (6)

Use method 2:

Employees(Emp#, Name, Age, C_name)

Employee-Hobby(Emp#, Hobby)

Managers(Manager-Emp#, Name, Age, Budget, C_name)

Manager-Hobby(Manager-Emp#, Hobby)

Programmers(Programmer-Emp#, Name, Age, Years_of_experience, C_name)

Programmer-Hobby(Programmer-Emp#, Hobby)

# Transform a Complex ER Diagram (7)

Cities(C_name, Population)

Projects(Proj#, Name)

Languages(L_name)

Work_on(Emp#, Proj#, Hours)

Manager-Work_on(Manager-Emp#, Proj#, Hours)

Programmer-Work_on(Programmer-Emp#, Proj#, Hours)

Use(Programmer-Emp#, Proj#, L_name)

# Transform a Complex ER Diagram (8)

- Employee-Hobby.Emp# and Work_on.Emp# are foreign keys referencing Employees.Emp#
- Manager-Hobby.Manager-Emp# and Manager-Work-on.Manager-Emp# are foreign keys referencing Managers.Manager-Emp#
- Programmer-Hobby.Programmer-Emp#, Programmer-Work-on.Programmer-Emp# and Use.Programmer-Emp# are foreign keys referencing Programmers.Programmer-Emp#
- Employees.C_name, Managers.C_name and Programmers.C_name are foreign keys referencing Cities.C_name
- Work_on.Proj#, Manager-Work-on.Proj#, Programmer-Work-on.Proj# and Use.Proj# are foreign keys referencing Projects.Proj#
- Use.L_name is a foreign key referencing Languages.L_name

# Transform a Complex ER Diagram (9)

Use method 3:

Employees(Emp#, Name, Age, C_name, Budget,
Years_of_programming_experience, Job_type)

Employee-Hobby(Emp#, Hobby)

Cities(C_name, Population)

Projects(Proj#, Name)

Languages(L_name)

Work_on(Emp#, Proj#, Hours)

Use(Emp#, Proj#, L_name)

# Transform a Complex ER Diagram (10)

Use method 3 (continued):

- Employee-Hobby.Emp# and Work_on.Emp# and Use.Emp# are foreign keys referencing Employees.Emp#
- Employees.C_name is a foreign key referencing Cities.C_name
- Work_on.Proj# and Use.Proj# are foreign keys referencing Projects.Proj#
- Use.L_name is a foreign key referencing Languages.L_name