International University, VNU-HCMC

School of Computer Science and Engineering

Lecture 3:
ER – Relational Translation

Instructor: Nguyen Thi Thuy Loan

nttloan@hcmiu.edu.vn, nthithuyloan@gmail.com
https://nttloan.wordpress.com/

Acknowledgement

Assoc. Prof. Nguyen Thi Thuy Loan, PhD

- The following slides have been created by adapting materials from the [GUW] book provided by the authors, Prof. Jeffrey D. Ullman and others.
- Other slides are referenced from Northeastern University and Duke University.

Purpose of the Lecture

Assoc. Prof. Nguyen Thi Thuy Loan, PhD

- To explain how to translate Entity–Relationship (ER) models into Relational Schemas.
- To provide a step-by-step method for mapping entities, attributes, and relationships.
- To ensure students can design relational databases that accurately represent real-world requirements.

Warm-up Question

Assoc. Prof. Nguyen Thi Thuy Loan, PhD

- Why do we need to translate an ER diagram into a relational model?

Outlines

- Introduction
- Mapping Rules Overview
- Detailed Mapping Cases
- Examples

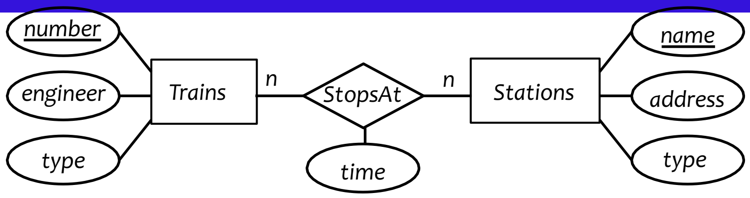
ER model: review

Design ERD

- Notation
- Entity/ Weak entity
- Attributes
- Relationship
  - ○ Attributes on relationships
  - ○ Multiplicity
  - ○ Binary versus n-ary relationships
  - ○ ISA relationships

Example

Assoc. Prof. Nguyen Thi Thuy Loan, PhD



You designed an ER digram

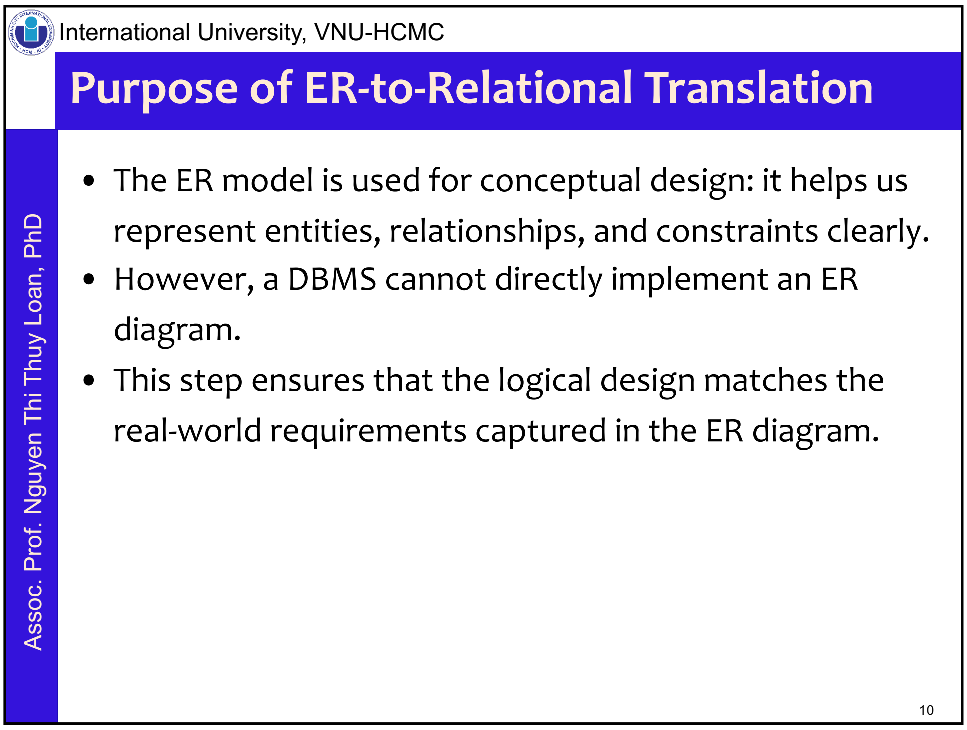
Translate it to a Relational Database

Train (number, engineer, type)
Station (name, address, type)
TrainStop (train_number, station_name, time)

Introduction

Assoc. Prof. Nguyen Thi Thuy Loan, PhD

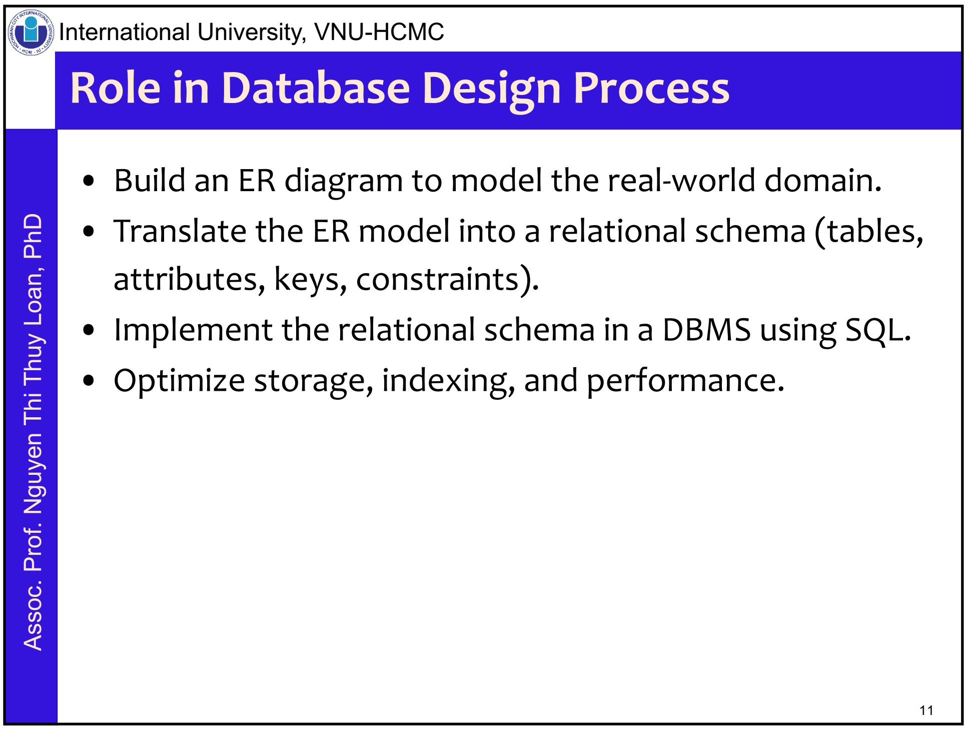
- Purpose of ER-to-Relational Translation
- Role in Database Design Process



Purpose of ER-to-Relational Translation

Assoc. Prof. Nguyen Thi Thuy Loan, PhD

- The ER model is used for conceptual design: it helps us represent entities, relationships, and constraints clearly.
- However, a DBMS cannot directly implement an ER diagram.
- This step ensures that the logical design matches the real-world requirements captured in the ER diagram.



Role in Database Design Process

Assoc. Prof. Nguyen Thi Thuy Loan, PhD

- Build an ER diagram to model the real-world domain.
- Translate the ER model into a relational schema (tables, attributes, keys, constraints).
- Implement the relational schema in a DBMS using SQL.
- Optimize storage, indexing, and performance.

Mapping Rules Overview

Step 1: Regular Entity Types

Create a relation (Table)

- For each strong entity type, make a table.
- Include all simple attributes (including simple attributes of composite relations).

Choose a primary key

- Pick one key attribute as the primary key.
- If the key is composite, use all its simple attributes together.

Keep other keys unique

- Any other key attributes become secondary unique keys.
- These are useful for indexing and faster queries.

Step 2: Weak Entity Types

Create a Relation (Table)

- For each weak entity, make a table.
- Include all its simple attributes.

Add Foreign Key

- Insert the primary key of the owner entity as a foreign key in this table.

Define Primary Key

- The primary key of the weak entity table is:
  - The owner's primary key (from the foreign key), plus
  - The partial key of the weak entity (if it exists).

Step 3: Mapping Binary 1-to-1

Choose One Relation as "S"

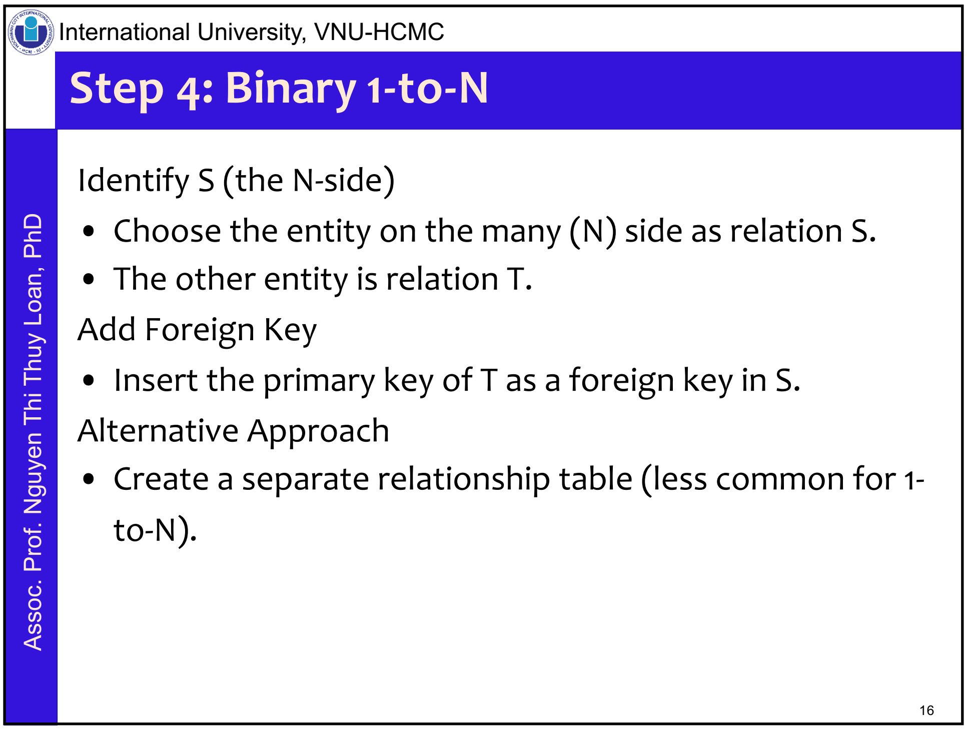
- The other relation is "T".
- Prefer S if it has total participation (reduces NULL values).

Add Relationship Attributes

- Put all the simple attributes of the relationship into S.

Add Foreign Key

- Insert the primary key of T as a foreign key in S.



Step 4: Binary 1-to-N

Identify S (the N-side)

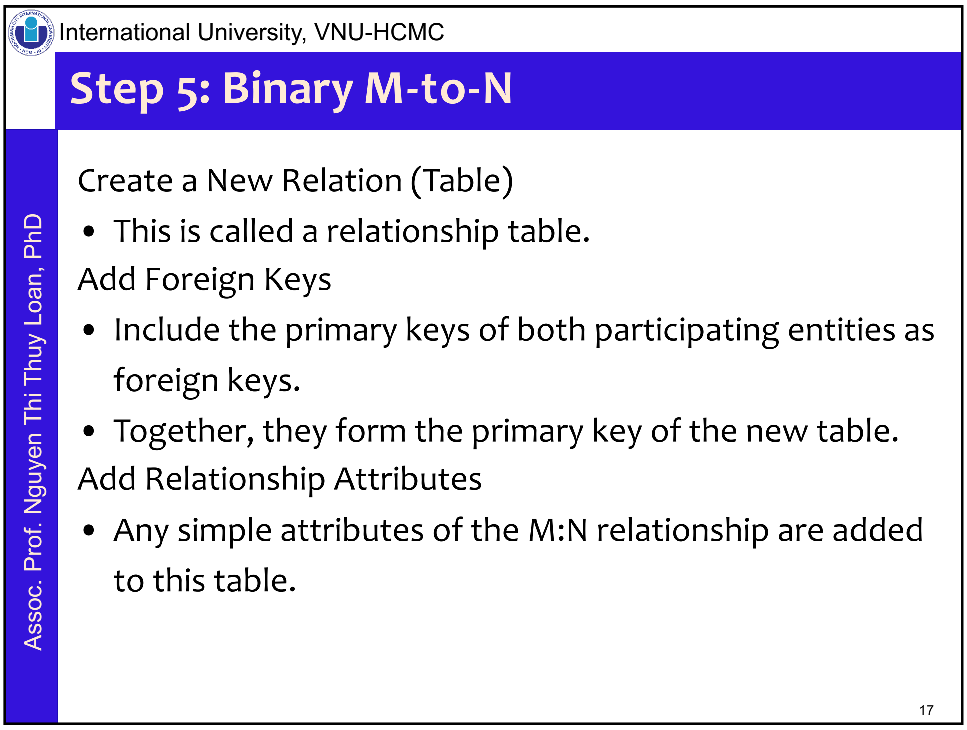
- Choose the entity on the many (N) side as relation S.
- The other entity is relation T.

Add Foreign Key

- Insert the primary key of T as a foreign key in S.

Alternative Approach

- Create a separate relationship table (less common for 1-to-N).



Step 5: Binary M-to-N

Create a New Relation (Table)

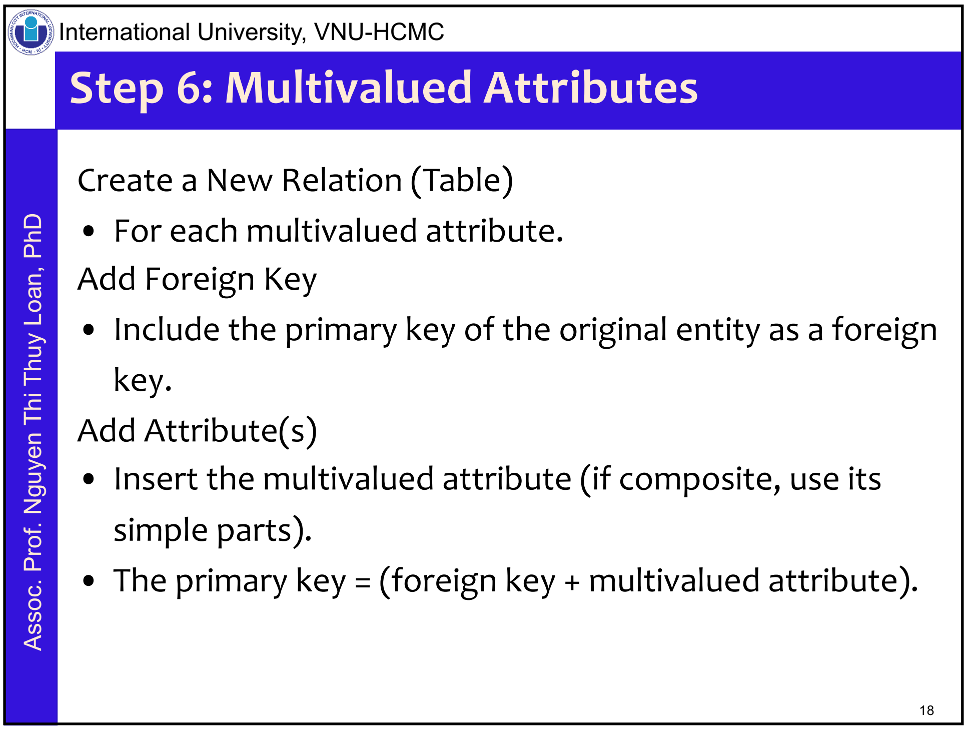
- This is called a relationship table.

Add Foreign Keys

- Include the primary keys of both participating entities as foreign keys.
- Together, they form the primary key of the new table.

Add Relationship Attributes

- Any simple attributes of the M:N relationship are added to this table.



Step 6: Multivalued Attributes

Create a New Relation (Table)

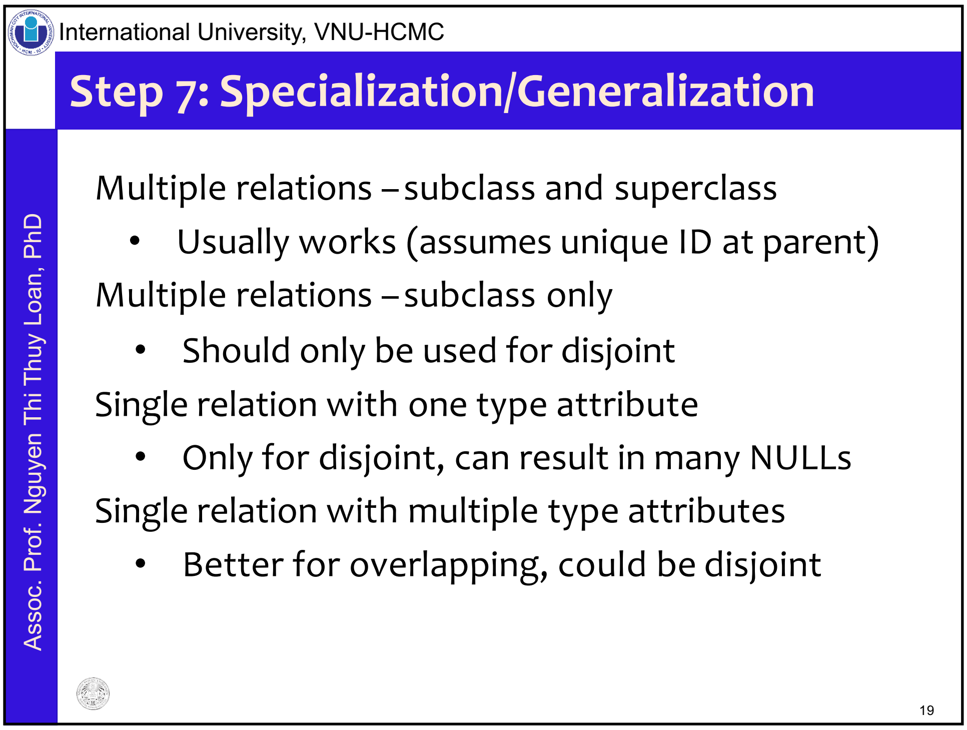
- For each multivalued attribute.

Add Foreign Key

- Include the primary key of the original entity as a foreign key.

Add Attribute(s)

- Insert the multivalued attribute (if composite, use its simple parts).
- The primary key = (foreign key + multivalued attribute).



Step 7: Specialization/Generalization

Multiple relations – subclass and superclass

- Usually works (assumes unique ID at parent)

Multiple relations – subclass only

- Should only be used for disjoint

Single relation with one type attribute

- Only for disjoint, can result in many NULLs

Single relation with multiple type attributes

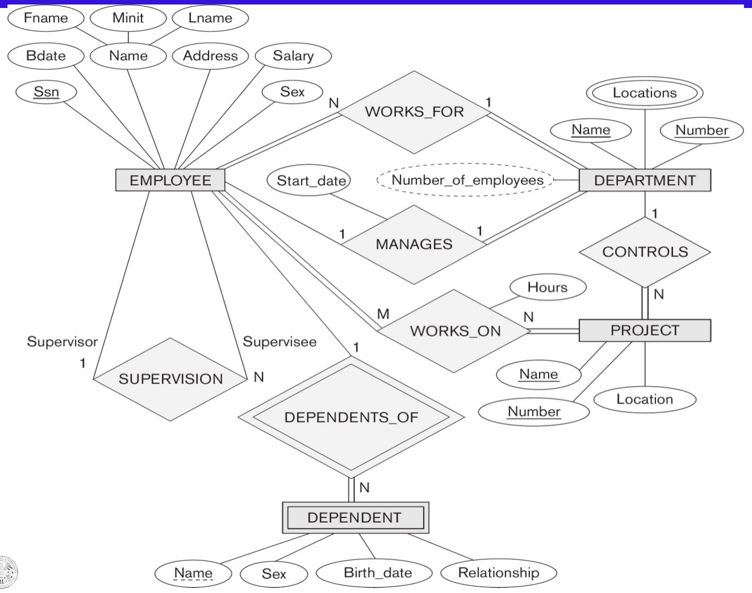
- Better for overlapping, could be disjoint

Assoc. Prof. Nguyen Thi Thuy Loan, PhD

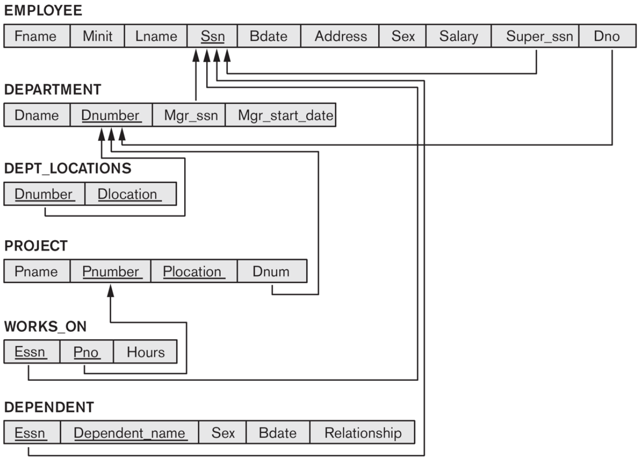
Detailed Mapping Cases

Assoc. Prof. Nguyen Thi Thuy Loan, PhD

Detailed Mapping Cases



Resulting Relational Schema



EMPLOYEE

Fname	Minit	Lname	Ssn	Bdate	Address	Sex	Salary	Super_ssn	Dno

DEPARTMENT

Dname	Dnumber	Mgr_ssn	Mgr_start_date

DEPT_LOCATIONS

Dnumber	Dlocation

PROJECT

Pname	Pnumber	Plocation	Dnum

WORKS_ON

Essn	Pno	Hours

DEPENDENT

Essn	Dependent_name	Sex	Bdate	Relationship

Step 1: Regular Entity Types

Create a relation (Table)

- For each strong entity type, make a table.
- Include all simple attributes (including simple attributes of composite relations).

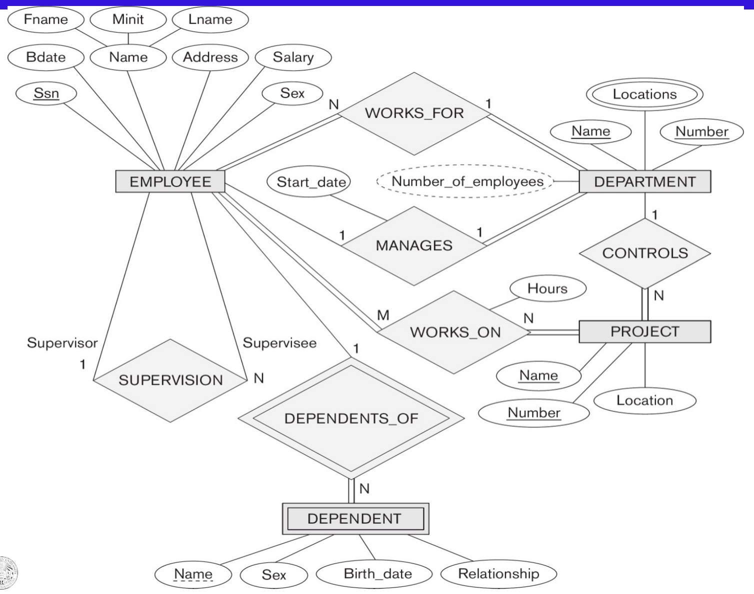
Choose a primary key

- Pick one key attribute as the primary key.
- If the key is composite, use all its simple attributes together.

Keep other keys unique

- Any other key attributes become secondary unique keys.
- These are useful for indexing and faster queries.

Detailed Mapping Cases



Step 1 Result

EMPLOYEE

Fname	Minit	Lmane	<u>Ssn</u>	Bdate	Address	Sex	Salary

DEPARTMENT

Dname	<u>Dnumber</u>

PROJECT

Pname	<u>Pnumber</u>	Plocation

Step 2: Weak Entity Types

Create a Relation (Table)

- For each weak entity, make a table.
- Include all its simple attributes.

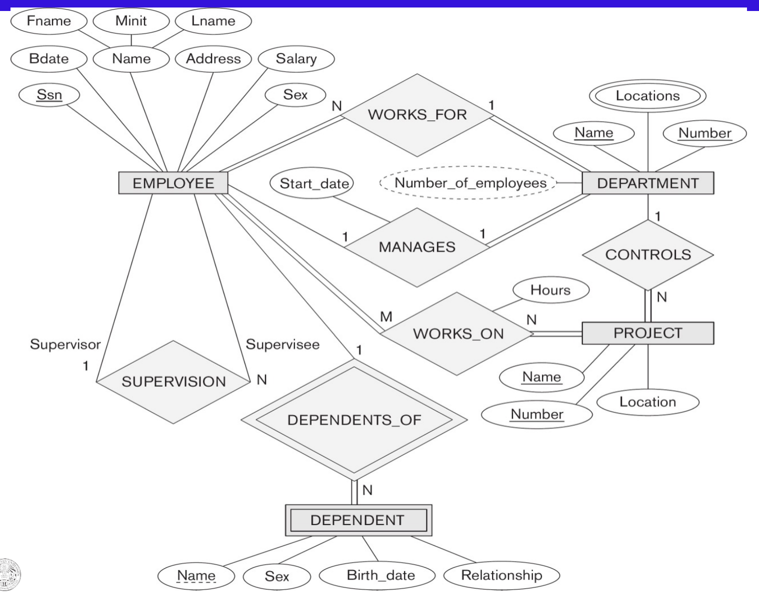
Add Foreign Key

- Insert the primary key of the owner entity as a foreign key in this table.

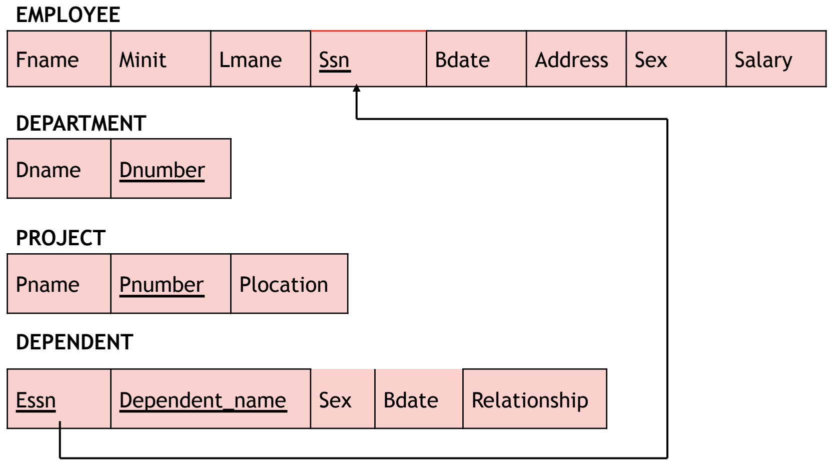
Define Primary Key

- The primary key of the weak entity table is:
  - The owner's primary key (from the foreign key), plus
  - The partial key of the weak entity (if it exists).

Detailed Mapping Cases



Step 2 Result



EMPLOYEE

Fname	Minit	Lmane	Ssn	Bdate	Address	Sex	Salary

DEPARTMENT

Dname	Dnumber

PROJECT

Pname	Pnumber	Plocation

DEPENDENT

Essn	Dependent_name	Sex	Bdate	Relationship

Step 3: Mapping Binary 1-to-1

Choose One Relation as “S”

- The other relation is “T”.
- Prefer S if it has total participation (reduces NULL values).

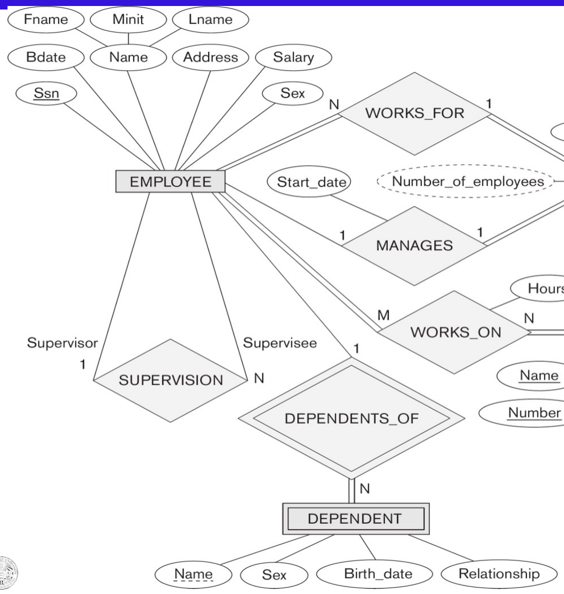
Add Relationship Attributes

- Put all the simple attributes of the relationship into S.

Add Foreign Key

- Insert the primary key of T as a foreign key in S.

Detailed Mapping Cases



Step 2 Result

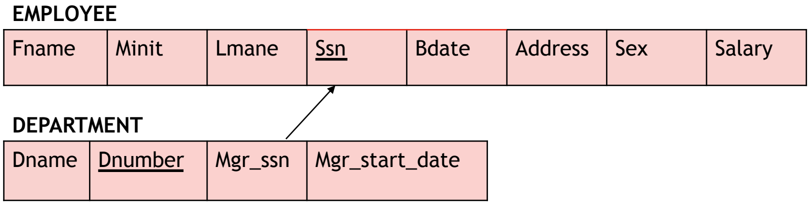
EMPLOYEE

Fname	Minit	Lname	<u>Ssn</u>	Bdate	Address	Sex	Salary

DEPARTMENT

Dname	<u>Dnumber</u>

Step 3 Result



EMPLOYEE

Fname	Minit	Lmane	Ssn	Bdate	Address	Sex	Salary

DEPARTMENT

Dname	Dnumber	Mgr_ssn	Mgr_start_date

Step 4: Binary 1-to-N

Identify S (the N-side)

- Choose the entity on the many (N) side as relation S.
- The other entity is relation T.

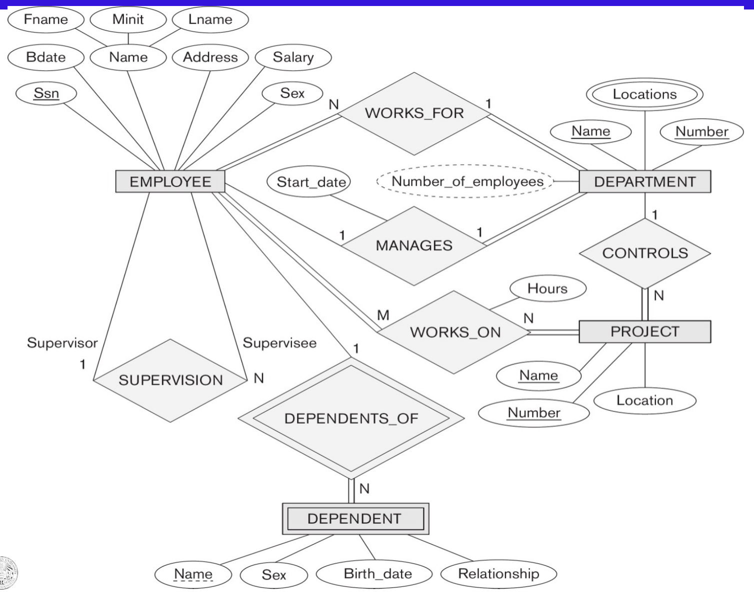
Add Foreign Key

- Insert the primary key of T as a foreign key in S.

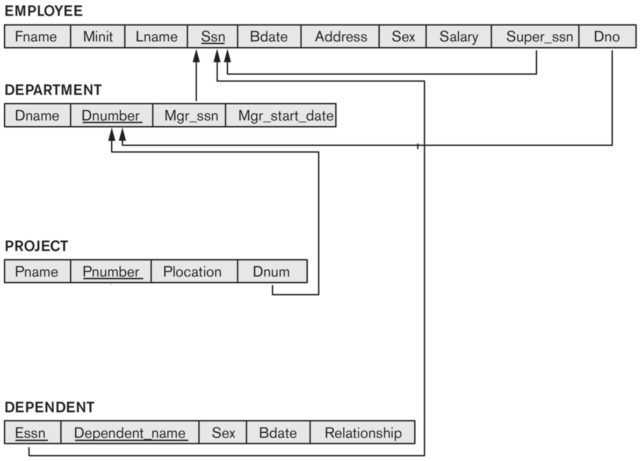
Alternative Approach

- Create a separate relationship table (less common for 1-to-N).

Detailed Mapping Cases



Step 4 Result



EMPLOYEE

Fname	Minit	Lname	Ssn	Bdate	Address	Sex	Salary	Super_ssn	Dno

DEPARTMENT

Dname	Dnumber	Mgr_ssn	Mgr_start_date

PROJECT

Pname	Pnumber	Plocation	Dnum

DEPENDENT

Essn	Dependent_name	Sex	Bdate	Relationship

Step 5: Binary M-to-N

Create a New Relation (Table)

- This is called a relationship table.

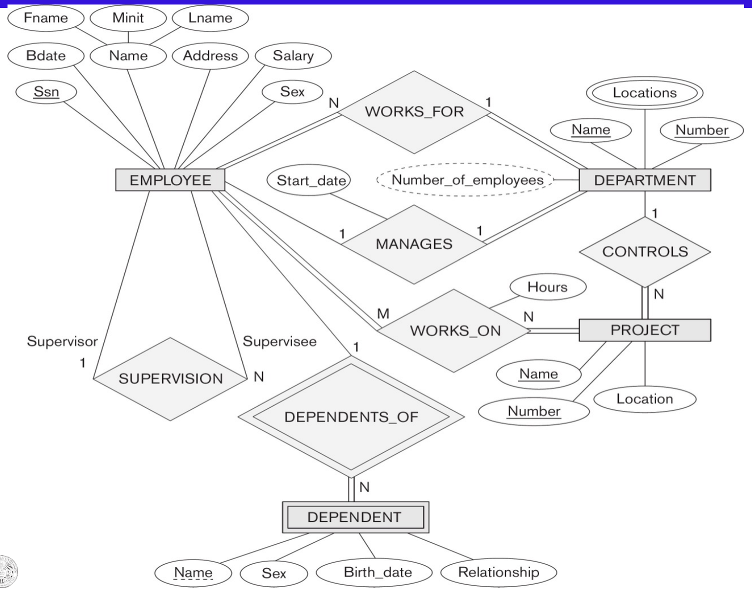
Add Foreign Keys

- Include the primary keys of both participating entities as foreign keys.
- Together, they form the primary key of the new table.

Add Relationship Attributes

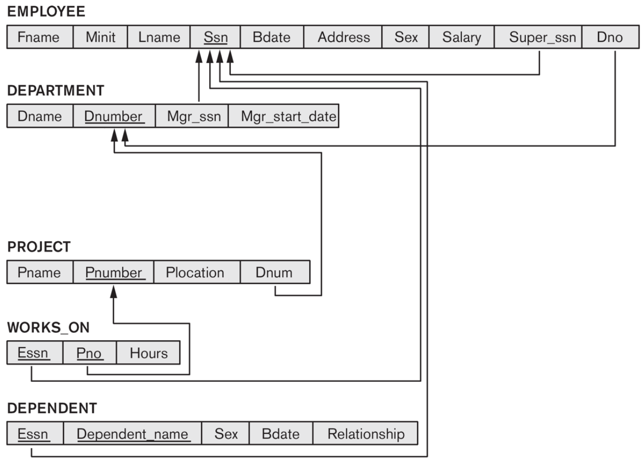
- Any simple attributes of the M:N relationship are added to this table.

Detailed Mapping Cases



Step 5 Result

Assoc. Prof. Nguyen Thi Thuy Loan, PhD



EMPLOYEE

Fname	Minit	Lname	Ssn	Bdate	Address	Sex	Salary	Super_ssn	Dno

DEPARTMENT

Dname	Dnumber	Mgr_ssn	Mgr_start_date

PROJECT

Pname	Pnumber	Plocation	Dnum

WORKS_ON

Essn	Pno	Hours

DEPENDENT

Essn	Dependent_name	Sex	Bdate	Relationship

Step 6: Multivalued Attributes

Assoc. Prof. Nguyen Thi Thuy Loan, PhD

Create a New Relation (Table)

- For each multivalued attribute.

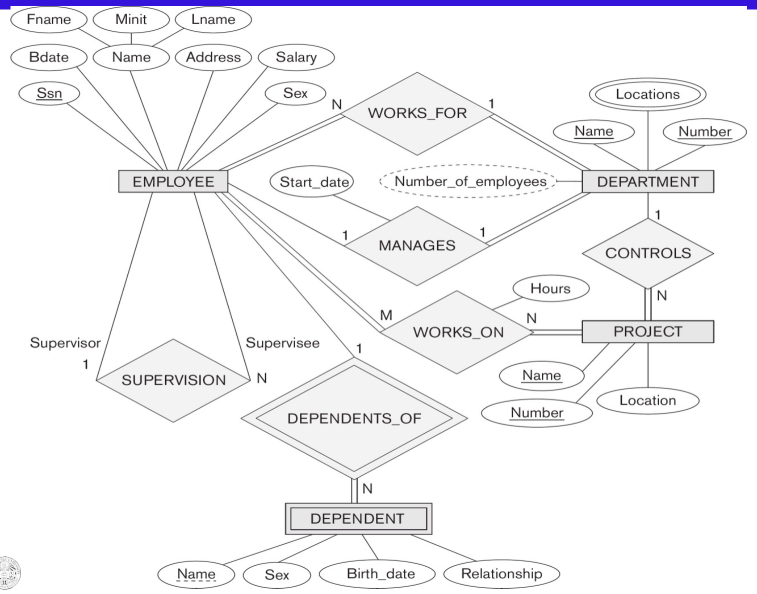
Add Foreign Key

- Include the primary key of the original entity as a foreign key.

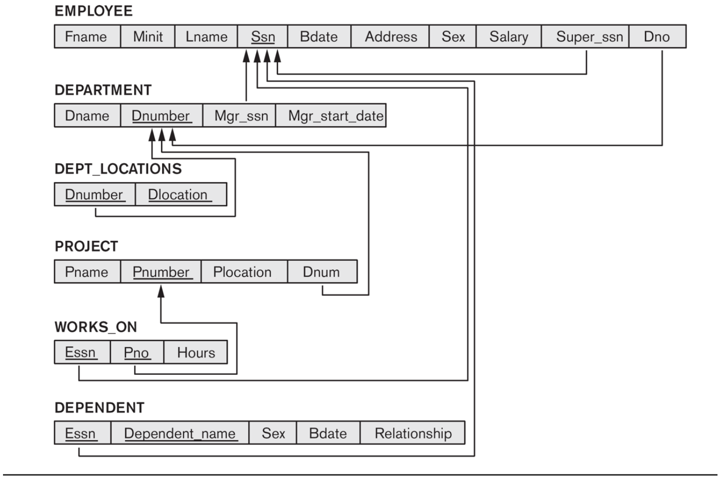
Add Attribute(s)

- Insert the multivalued attribute (if composite, use its simple parts).
- The primary key = (foreign key + multivalued attribute).

Detailed Mapping Cases



Step 6 Result



EMPLOYEE

Fname	Minit	Lname	Ssn	Bdate	Address	Sex	Salary	Super_ssn	Dno

DEPARTMENT

Dname	Dnumber	Mgr_ssn	Mgr_start_date

DEPT_LOCATIONS

Dnumber	Dlocation

PROJECT

Pname	Pnumber	Plocation	Dnum

WORKS_ON

Essn	Pno	Hours

DEPENDENT

Essn	Dependent_name	Sex	Bdate	Relationship

Step 7: Specialization/Generalization

Multiple relations – subclass and superclass

- Usually works (assumes unique ID at parent)

Multiple relations – subclass only

- Should only be used for disjoint

Single relation with one type attribute

- Only for disjoint, can result in many NULLs

Single relation with multiple type attributes

- Better for overlapping, could be disjoint

Specialization/Generalization (A)

Multiple relations – subclass and superclass

Usually works (assumes unique ID at parent)

(a) EMPLOYEE

Ssn	Fname	Minit	Lname	Birth_date	Address	Job_type

SECRETARY

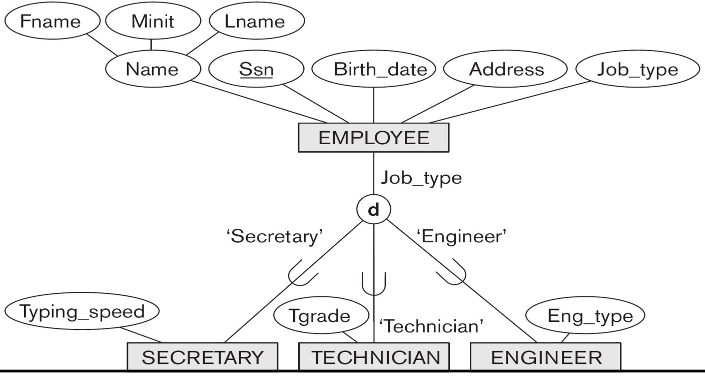
Ssn	Typing_speed

TECHNICIAN

Ssn	Tgrade

ENGINEER

Ssn	Eng_type



Specialization/Generalization (B)

Multiple relations –subclass only

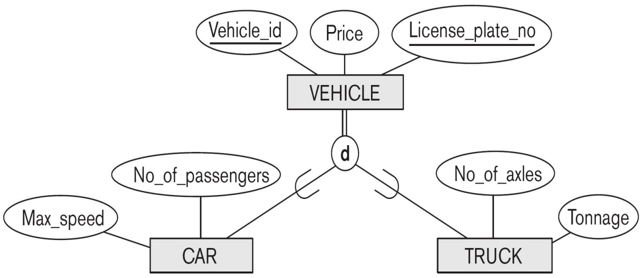
Should only be used for disjoint

CAR

Vehicle_id	License_plate_no	Price	Max_speed	No_of_passengers

TRUCK

Vehicle_id	License_plate_no	Price	No_of_axles	Tonnage



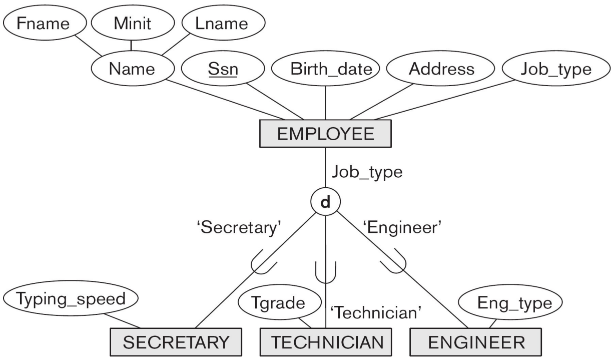
Specialization/Generalization (C)

Single relation with one type attribute

Only for disjoint, can result in many NULLs

EMPLOYEE

Ssn	Fname	Minit	Lname	Birth_date	Address	Job_type	Typing_speed	Tgrade	Eng_type



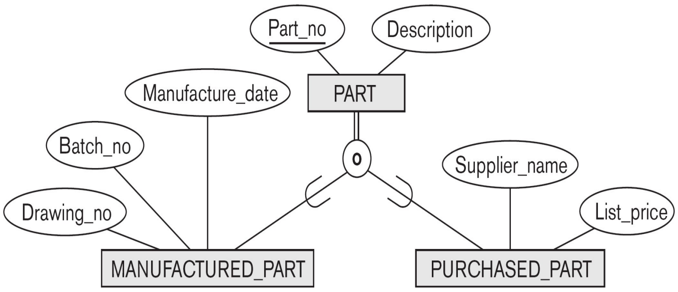
Specialization/Generalization (D)

Single relation with multiple type attributes

Better for overlapping, could be disjoint

PART

Part_no	Description	Mflag	Drawing_no	Manufacture_date	Batch_no	Pflag	Supplier_name	List_price



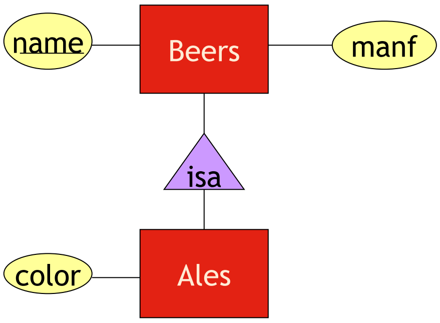
Assoc. Prof. Nguyen Thi Thuy Loan, PhD

Subclasses: Three Approaches

1. *Object-oriented*: One relation per subset of subclasses, with all relevant attributes.
2. *Use nulls*: One relation; entities have NULL in attributes that don’t belong to them.
3. *E/R style*: One relation for each subclass:
   - Key attribute(s).
   - Attributes of that subclass.

Assoc. Prof. Nguyen Thi Thuy Loan, PhD

Example: Subclass → Relations



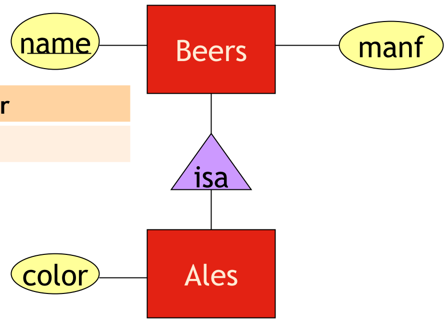
Object-Oriented

Name	Manf
Bud	Anheuser-Busch

Beers

Name	Manf	Color
Summerbrew	Pete's	Drak

Ales



Good for queries like "find the color of ales made by Pete's."

E/R Style

Name	Manf
Bud	Anheuser-Busch
Summerbrew	Pete’s

Beers

Name	Color
Summerbrew	Drak

Ales

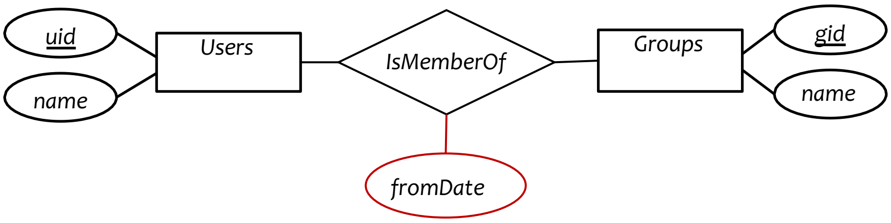
Good for queries like “find all beers (including ales) made by Pete’s.”

Using Nulls

Name	Manf	Color
Bud	Anheuser-Busch	Null
Summerbrew	Pete’s	Drak

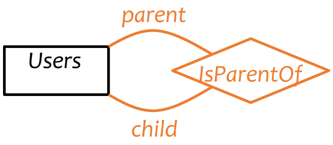
Saves space unless there are *lots* of attributes that are usually NULL.

Examples

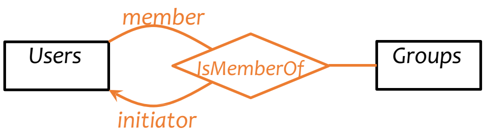


- Where do the dates go?
  - With *Users*?
    - But a user can join multiple groups on different dates
  - With *Groups*?
    - But different users can join the same group on different dates
  - With *IsMemberOf*!

More examples



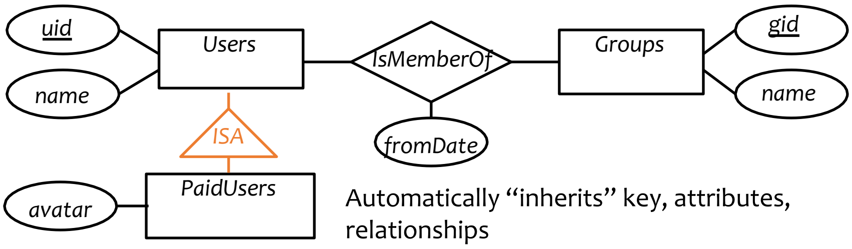
Parent (parent_uid, child_uid)



Member (uid, initiator_uid, gid)

ISA relationships

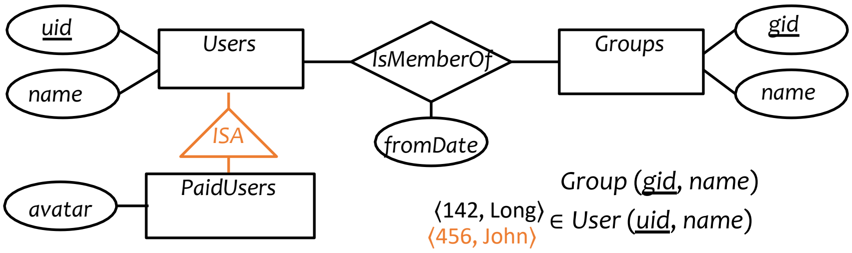
- Similar to the idea of subclasses in object-oriented programming: subclass = special case, fewer entities, and possibly more properties
  - Represented as a triangle (direction is important)
- Example: paid users are users, but they also get avatars (yay!)



Automatically "inherits" key, attributes, relationships

Translating subclasses & ISA: approach 1

- Entity-in-all-superclasses approach ("E/R style")
  - An entity is represented in the table for each subclass to which it belongs
  - A table includes only the attributes directly attached to the corresponding entity set, plus the inherited key



Group (<u>gid</u>, name)

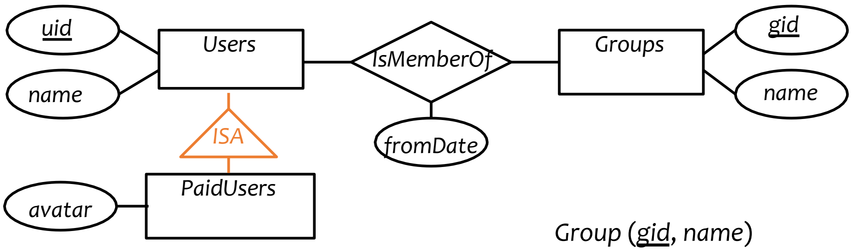
⟨142, Long⟩ ⟨456, John⟩ ∈ *User (<u>uid</u>, name)*

Member (<u>uid</u>, <u>gid</u>, from_date)

⟨456, ☺ ⟩ ∈ *PaidUser (<u>uid</u>, avatar)*

Translating subclasses & ISA: approach 2

- Entity-in-most-specific-class approach (“OO style”)
  - An entity is only represented in one table (the most specific entity set to which the entity belongs)
  - A table includes the attributes attached to the corresponding entity set, plus all inherited attributes



Group (gid, name)

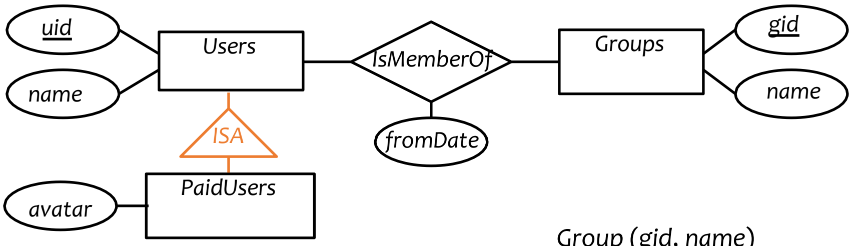
⟨142, Long⟩ ∈ User (uid, name)

Member (uid, gid, from_date)

⟨456, Ralph, ☺⟩ ∈ PaidUser (uid, name, avatar)

Translating subclasses & ISA: approach 3

- All-entities-in-one-table approach (“NULL style”)
  - One relation for the root entity set, with all attributes found in the network of subclasses (plus a “type” attribute when needed)
  - Use a special NULL value in columns that are not relevant for a particular entity

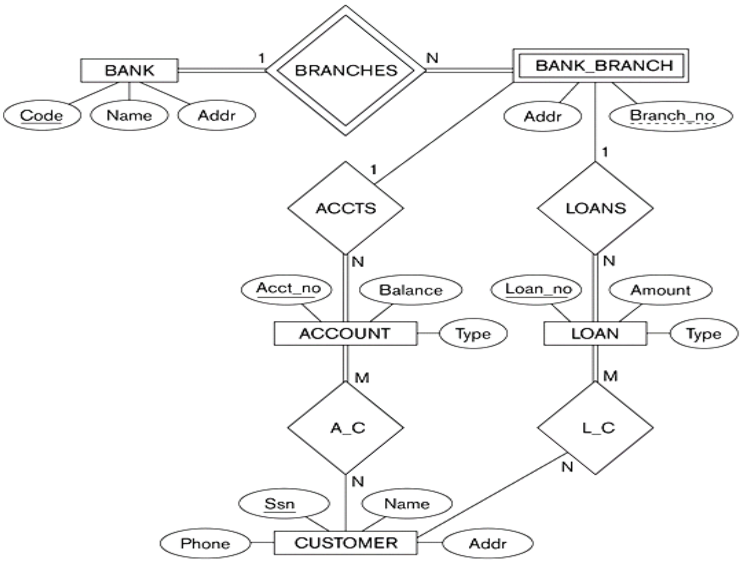


Group (gid, name)

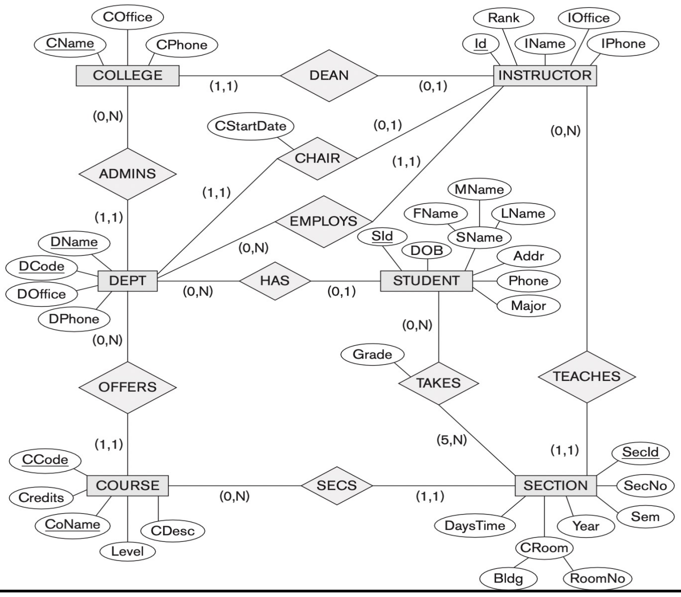
⟨142, Long , NULL⟩
⟨456, John, ☺ ⟩ ∈ User (uid, name, avatar)

Member (uid, gid, from_date)

Example ERD



Example ERD



Thank you for your attention!