

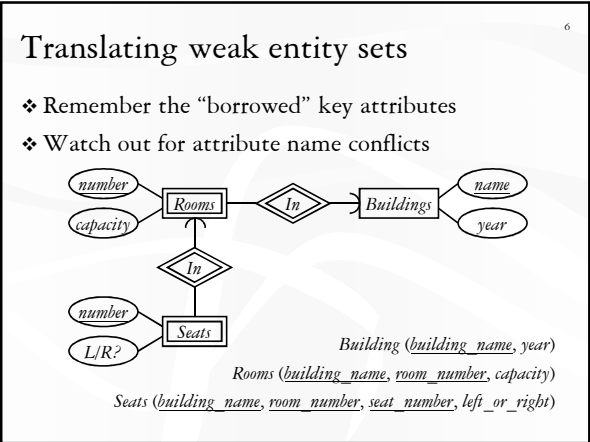
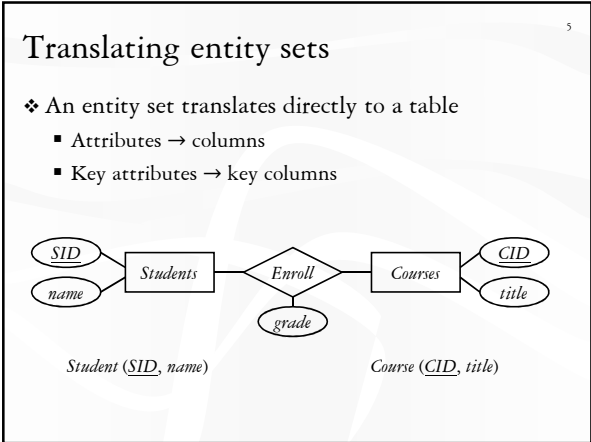
# Relational Database Design Part II

CompSci 316  
Introduction to Database Systems

- ## Announcements (Tue. Sep. 10)
- ❖ Homework #1 is due in a week
  - ❖ Project is now officially “assigned”
    - See “Assignments” on course website for handout
    - Let me know if there is anything I can do to help you form project ideas/groups
  - ❖ Interesting project ideas in upcoming talk
    - Derek Willis from *The New York Times*
    - 11:45am-12:45pm, Monday Oct. 7; Perkins LINK 5
  - ❖ Anonymous feedback is welcome
    - See course website for link
  - ❖ Interested in Piazza?

- ## Database design steps: review
- ❖ Understand the real-world domain being modeled
  - ❖ Specify it using a database design model (e.g., E/R)
  - ❖ Translate specification to the data model of DBMS (e.g., relational)
  - ❖ Create DBMS schema
- ☞ Next: translating E/R design to relational schema

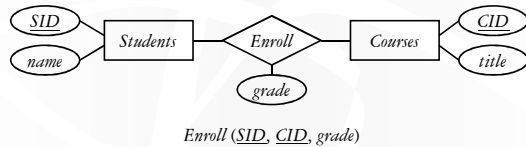
- ## E/R model: review
- ❖ Entity sets
    - Keys
    - Weak entity sets
  - ❖ Relationship sets
    - Attributes on relationships
    - Multiplicity
    - Roles
    - Binary versus *N*-ary relationships
      - Modeling *N*-ary relationships with weak entity sets and binary relationships
    - ISA relationships



## Translating relationship sets

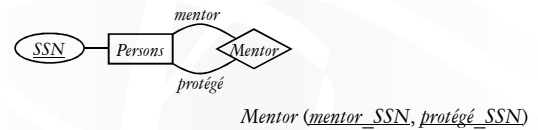
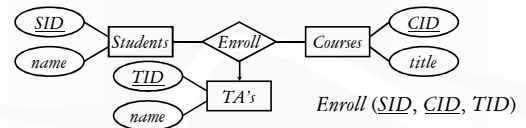
7

- ❖ A relationship set translates to a table
  - Keys of connected entity sets → columns
  - Attributes of the relationship set (if any) → columns
  - Multiplicity of the relationship set determines the key of the table



## More examples

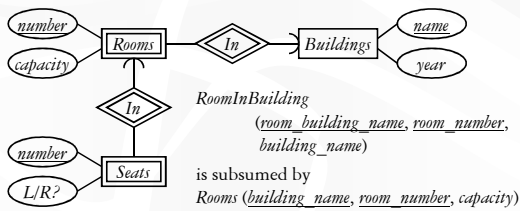
8



## Translating double diamonds

9

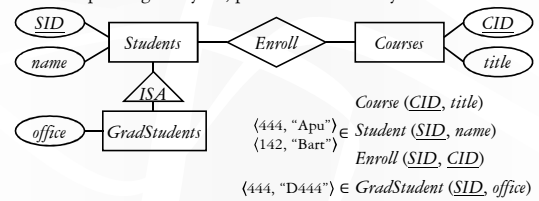
- ❖ Recall that a double-diamond (supporting) relationship set connects a weak entity set to another entity set
- ❖ No need to translate because the relationship is implicit in the weak entity set's translation



## Translating subclasses & ISA (approach 1)

10

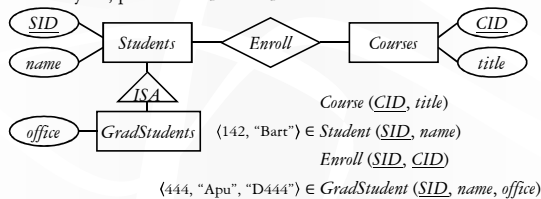
- ❖ Entity-in-all-superclasses approach ("E/R style")
  - An entity is represented in the table for each subclass to which it belongs
  - A table includes only the attributes directly attached to the corresponding entity set, plus the inherited key



## Translating subclasses & ISA (approach 2)

11

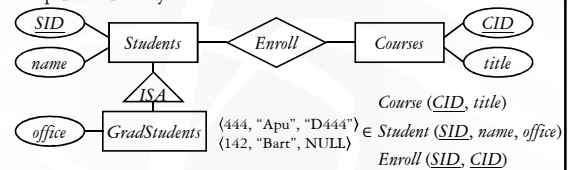
- ❖ Entity-in-most-specific-class approach ("OO style")
  - An entity is only represented in one table (corresponding to the most specific entity set to which the entity belongs)
  - A table includes the attributes attached to the corresponding entity set, plus all inherited attributes



## Translating subclasses & ISA (approach 3)

12

- ❖ 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



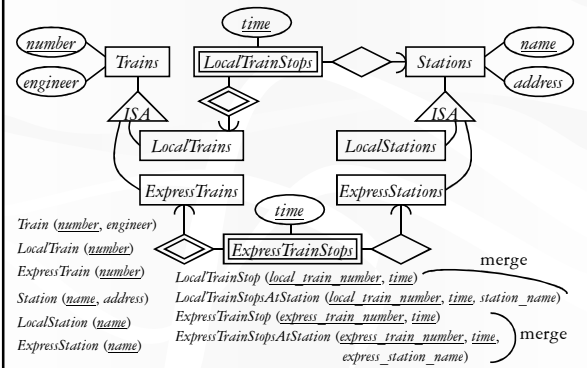
## Comparison of three approaches

13

- ❖ Entity-in-all-superclasses
  - *Student* (SID, name), *GradStudent* (SID, office)
  - Pro: All students are found in one table
  - Con: Attributes of grad students are scattered in different tables
- ❖ Entity-in-most-specific-class
  - *Student* (SID, name), *GradStudent* (SID, name, office)
  - Pro: All attributes of grad students are found in one table
  - Con: Students are scattered in different tables
- ❖ All-entities-in-one-table
  - *Student* (SID, [type, ]name, office)
  - Pro: Everything is in one table
  - Con: Too many NULL's; complicated if class hierarchy is complex

## A complete example

14



## Simplifications and refinements

15

*Train* (number, engineer, type)  
*Station* (name, address, type)  
*LocalTrainStop* (local\_train\_number, station\_name, time)  
*ExpressTrainStop* (express\_train\_number, express\_station\_name, time)

- ❖ Eliminate *LocalTrain* table
  - Redundant: can be computed as  $\pi_{\text{number}}(\text{Train}) - \text{ExpressTrain}$
  - Slightly harder to check that *local\_train\_number* is indeed a local train number
- ❖ Eliminate *LocalStation* table
  - It can be computed as  $\pi_{\text{number}}(\text{Station}) - \text{ExpressStation}$

## An alternative design

16

*Train* (number, engineer, type)  
*Station* (name, address, type)  
*TrainStop* (train\_number, station\_name, time)

- ❖ Encode the type of train/station as a column rather than creating subclasses
- ❖ What about the following constraints?
  - Type must be either "local" or "express"
  - Express trains only stop at express stations
  - ☞ They can be expressed/declared explicitly as database constraints in SQL (as we will see later in course)
- ❖ Arguably a better design because it is simpler!

## Design principles

17

- ❖ KISS
  - Keep It Simple, Stupid
- ❖ Avoid redundancy
  - Redundancy wastes space, complicates modifications, promotes inconsistency
- ❖ Capture essential constraints, but don't introduce unnecessary restrictions
- ❖ Use your common sense
  - Warning: mechanical translation procedures given in this lecture are no substitute for your own judgment