Relational Database Design Part II

CPS 116

Introduction to Database Systems

Announcements (Thu. Sep. 3)

- Homework #1 due in 1½ weeks
  - Start early!!!
- Details of the course project and a list of suggested ideas will be available next Thursday

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

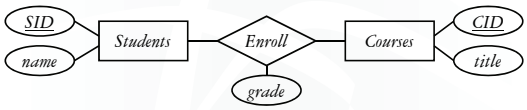
☞ Next: translating E/R design to relational schema

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 entity sets

- An entity set translates directly to a table
  - Attributes → columns
  - Key attributes → key columns

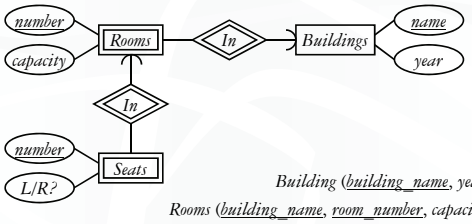


Student (*SID*, *name*)

Course (*CID*, *title*)

Translating weak entity sets

- Remember the "borrowed" key attributes
- Watch out for attribute name conflicts



Building (*building_name*, *year*)

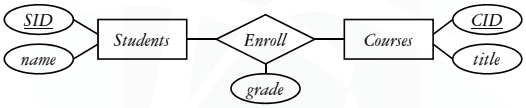
Rooms (*building_name*, *room_number*, *capacity*)

Seats (*building_name*, *room_number*, *seat_number*, *left_or_right*)

Translating relationship sets

7

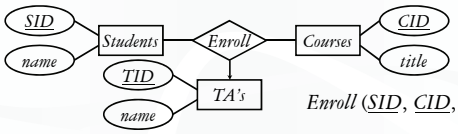
- A relationship set translates to a table
  - Keys of connected entity sets → columns
  - Attributes of the relationship set (if any) → columns
  - Multiplicity of the relationship set determines the key of the table



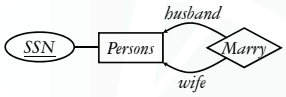
Enroll (*SID*, *CID*, *grade*)

More examples

8



Enroll (*SID*, *CID*, *TID*)

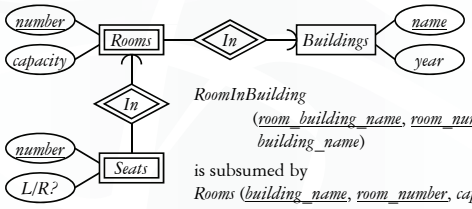


Marry (*husband_SSN*, *wife_SSN*)
Marry (*husband_SSN*, *wife_SSN*)

Translating double diamonds

9

- Recall that a double-diamond 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



RoomInBuilding (*room_building_name*, *room_number*, *building_name*)

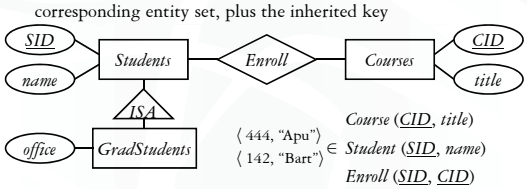
is subsumed by

Rooms (*building_name*, *room_number*, *capacity*)

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



Course (*CID*, *title*)

⟨ 444, "Apu" ⟩ ∈ *Student* (*SID*, *name*)
⟨ 142, "Bart" ⟩

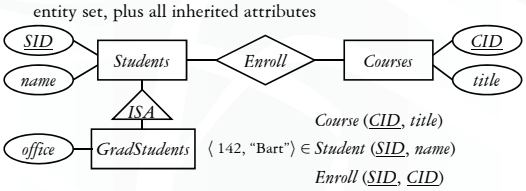
Enroll (*SID*, *CID*)

⟨ 444, "D444" ⟩ ∈ *GradStudent* (*SID*, *office*)

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



Course (*CID*, *title*)

⟨ 142, "Bart" ⟩ ∈ *Student* (*SID*, *name*)

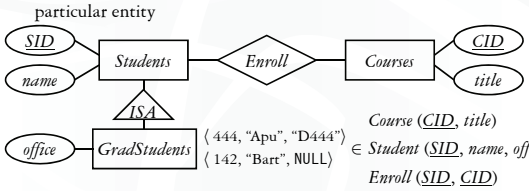
Enroll (*SID*, *CID*)

⟨ 444, "Apu", "D444" ⟩ ∈ *GradStudent* (*SID*, *name*, *office*)

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 anywhere in the network of subclasses
  - Use a special NULL value in columns that are not relevant for a particular entity



Course (*CID*, *title*)

⟨ 444, "Apu", "D444" ⟩
⟨ 142, "Bart", NULL ⟩ ∈ *Student* (*SID*, *name*, *office*)

Enroll (*SID*, *CID*)

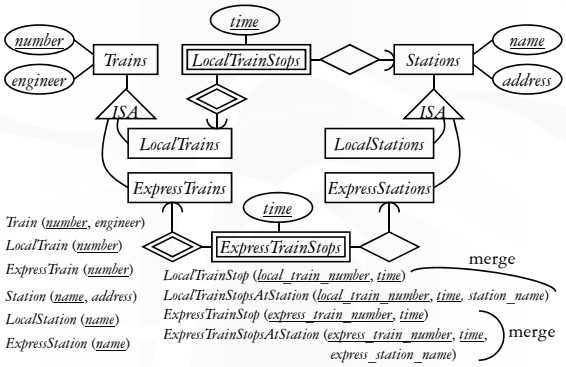
Comparison of three approaches

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*, *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*), *LocalTrain* (*number*), *ExpressTrain* (*number*)
Station (*name*, *address*), *LocalStation* (*name*), *ExpressStation* (*name*)
LocalTrainStop (*local_train_number*, *station_name*, *time*)
ExpressTrainStop (*express_train_number*, *express_station_name*, *time*)

- Eliminate *LocalTrain* table
  - Redundant: can be computed as
    $$\pi_{number}(Train) - ExpressTrain$$
    - ☞Why is redundancy bad?
  - Slightly harder to check that *local_train_number* is indeed a local train number
- Eliminate *LocalStation* table
  - It can be computed as $\pi_{number}(Station) - ExpressStation$

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
- Some constraints are no longer captured
  - Type must be either "local" or "express"
  - Express trains only stop at express stations
  - ☞Fortunately, they can be expressed/declared explicitly as database constraints in SQL
- ☞ Arguably a better design because it is simpler!

Design principles

17

- KISS
  - Keep It Simple, Stupid
- Avoid redundancy
  - Redundancy wastes space, complicates updates and deletes, 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