Announcements (Thu. Sep. 6)

- Homework #1 due in 1.5 weeks
  - By the end of this lecture, you should be able to complete Problems 1-3, 4(a), X1, X2
- Course project description available this week
  - Choice of “standard” or “open”
  - Team of 1-4, but single-person projects need approval
  - Two milestones + demo/report
- Milestone #1 due in ~one month, right after fall break

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

Translating weak entity sets

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

[Diagram of entity and relationship sets]
Translating relationship sets

- 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

Students

Courses

Enroll (SID, CID, grade)

More examples

Enroll (SID, CID, TID)

Mentor (mentor SSN, protégé SSN)

Translating double diamonds

- 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

Rooms

Buildings

RoomInBuilding (room building name, room number, building name)

is subsumed by Rooms (building name, room number, capacity)

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

Students

Courses

Enroll (SID, CID, title)

GradStudents

ISA Course (CID, title)

Student (SID, name)

Enroll (SID, CID)

GradStudent (SID, name, office)

Translating subclasses & ISA (approach 2)

- 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

Students

Courses

Enroll (SID, CID, title)

GradStudents

{444, "Apu"} ∈ GradStudent (SID, name, office)

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

Students

Courses

Enroll (SID, CID, title)

GradStudents

{444, "Apu", "D444"} ∈ GradStudent (SID, name, office)
Comparison of three approaches

- **Entity-in-all-superclasses**
  - Student (SID, name), GradStudent (SID, office)
  - **Pros**: All students are found in one table
  - **Cons**: Attributes of grad students are scattered in different tables

- **Entity-in-most-specific-class**
  - Student (SID, name), GradStudent (SID, name, office)
  - **Pros**: All attributes of grad students are found in one table
  - **Cons**: Students are scattered in different tables

- **All-entities-in-one-table**
  - Student (SID, name, office), GradStudent (SID, name, office)
  - **Pros**: All attributes of grad students are found in one table
  - **Cons**: Students are scattered in different tables

Simplifications and refinements

- 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 \( \pi_{\text{number}}(\text{Train}) – \text{ExpressTrain} \)
  - 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

- 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)

Design principles

- **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