Relational Database Design:
E/R-Relational Translation

Introduction to Databases
CompSci 316 Fall 2016

Announcements (Tue. Sep. 13)
• Homework #1 due in one week
  • Please please please start early
• Project description available soon

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
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

\[
\begin{align*}
\text{Users} & \rightarrow \text{name, uid} \\
\text{Groups} & \rightarrow \text{name, gid} \\
\text{Member} & \rightarrow \text{uid, gid, fromDate}
\end{align*}
\]

More examples

\[
\begin{align*}
\text{Users} & \rightarrow \text{name, uid} \\
\text{Groups} & \rightarrow \text{name, gid} \\
\text{Parent} & \rightarrow \text{parent uid, child uid} \\
\text{Member} & \rightarrow \text{uid, initiator uid, gid}
\end{align*}
\]

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

\[
\begin{align*}
\text{Rooms} & \rightarrow \text{number, capacity} \\
\text{Buildings} & \rightarrow \text{name, year} \\
\text{RoomInBuilding} & \rightarrow \text{room building name, room number, building name} \\
\text{is subsumed by Room} & \rightarrow \text{building name, room number, capacity}
\end{align*}
\]
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

  ![Diagram of approach 1](image1.png)

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

  ![Diagram of approach 2](image2.png)

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

  ![Diagram of approach 3](image3.png)
Comparison of three approaches

- Entity-in-all-superclasses
  - User (uid, name), PaidUser (uid, avatar)
  - Pro: All users are found in one table
  - Con:

- Entity-in-most-specific-class
  - User (uid, name), PaidUser (uid, name, avatar)
  - Pro:
  - Con:

- All-entities-in-one-table
  - User (uid, [type, name, avatar])
  - Pro:
  - Con:

A complete example

![Diagram of a complete example]

Simplifications and refinements

- 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 $\pi_{\text{name}}(\text{Station}) = \text{ExpressStation}$
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)
- Arguably a better design because it is simpler!

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