Relational Database Design: E/R-Relational Translation

Introduction to Databases
CompSci 316 Fall 2017
Announcements (Tue. Sep. 12)

• 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 $\rightarrow$ columns
  • Key attributes $\rightarrow$ key columns

User (uid, name)        Group (gid, name)

- Users
  - uid
  - name

- Groups
  - gid
  - name

IsMemberOf

(fromDate)
Translating weak entity sets

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

**Entity Diagram**

- **Rooms**
  - `number`
  - `capacity`

- **Buildings**
  - `name`
  - `year`

- **Seats**
  - `number`
  - `L/R?`

**Entity Definitions**

- Building `(name, year)`
- Room `(building_name, room_number, capacity)`
- Seat `(building_name, room_number, seat_number, left_or_right)`
Translating relationship sets

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

$$
\text{Member} \ (\text{uid}, \text{gid}, \text{fromDate})
$$
More examples

Parent (parent_uid, child_uid)

Member (uid, initiator_uid, gid)
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
  number  capacity

 rooms_in_building  (room_building_name, room_number, building_name)

 seats
  number  L/R?

 room (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

![Entity-Relationship Diagram](image)

- Group (gid, name)
- Member (uid, gid, from_date)
- PaidUser (uid, 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

```
User (uid, name)
Group (gid, name)
Member (uid, gid, from_date)
PaidUser (uid, name, avatar)
```

Diagram:
- Users (uid, name)
- Groups (gid, name)
- PaidUsers (uid, name, avatar)
- ISA relationships:
  - Users ISA PaidUsers
  - Groups ISA Group
- IsMemberOf relationship:
  - Users IsMemberOf Group

Examples:
- \(\langle 142, \text{Bart} \rangle \in \text{User} (\text{uid}, \text{name})\)
- \(\langle 456, \text{Ralph}, \text{☺} \rangle \in \text{PaidUser} (\text{uid}, \text{name}, \text{avatar})\)
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

```sql
table Users
  (uid, name)
table Groups
  (gid, name)
table PaidUsers
  (uid, name, avatar)

// Example data
User (142, Bart, NULL)
User (456, Ralph, 😊)
Group (gid, name)
Member (uid, gid, from_date)
```
Comparison of three approaches

• Entity-in-all-superclasses
  • User \((uid, \text{name})\), PaidUser \((uid, \text{avatar})\)
  • Pro:
  • Con:

• Entity-in-most-specific-class
  • User \((uid, \text{name})\), PaidUser \((uid, \text{name, avatar})\)
  • Pro:
  • Con:

• All-entities-in-one-table
  • User \((uid, [\text{type, } \text{name, avatar}])\)
  • Pro:
  • Con:
A complete example

Train (number, engineer)
LocalTrain (number)
ExpressTrain (number)

Station (name, address)
LocalStation (name)
ExpressStation (name)

LocalTrainStop (local_train_number, time)
LocalTrainStopsAtStation (local_train_number, time, station_name)
ExpressTrainStop (express_train_number, time)
ExpressTrainStopsAtStation (express_train_number, time, express_station_name)
Simplifications and refinements

Train \((\text{number, engineer})\), LocalTrain \((\text{number})\), ExpressTrain \((\text{number})\)
Station \((\text{name, address})\), LocalStation \((\text{name})\), ExpressStation \((\text{name})\)
LocalTrainStop \((\text{local\_train\_number, station\_name, time})\)
ExpressTrainStop \((\text{express\_train\_number, express\_station\_name, time})\)

• Eliminate \text{LocalTrain} table
  • Redundant: can be computed as 
    \[ \pi_{\text{number}}(\text{Train}) - \text{ExpressTrain} \]
  • Slightly harder to check that \text{local\_train\_number} is indeed a local train number

• Eliminate \text{LocalStation} table
  • It can be computed as \[ \pi_{\text{number}}(\text{Station}) - \text{ExpressStation} \]
An alternative design

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

• 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