Relational Database Design:
E/R-Relational Translation

Introduction to Databases
CompSci 316 Fall 2014
Announcements (Tue. Sep. 9)

• Homework #1 due next Tuesday
• Project description available this Thursday
• Homework #2 to be assigned next Tuesday

• Office hours posted
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

User (uid, name)  Group (gid, name)
Translating weak entity sets

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

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 → columns
  - Attributes of the relationship set (if any) → 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.

RoomInBuilding
(room_building_name, room_number, building_name)

is subsumed by
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

\[
\begin{align*}
\text{Users} & \quad \text{Groups} \\
\text{uid} & \quad \text{gid} \\
\text{name} & \quad \text{name} \\
\text{ ISA} & \\
\text{PaidUsers} & \\
\text{avatar} & \\
\end{align*}
\]

\[
\{142, \text{Bart}\} \in \text{User (uid, name)} \\
\{456, \text{Ralph}\} \in \text{Member (uid, gid, from_date)} \\
\{456, \text{☺}\} \in \text{PaidUser (uid, avatar)} \\
\]

Group (gid, name)
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

```plaintext
<table>
<thead>
<tr>
<th>Users</th>
<th>Groups</th>
</tr>
</thead>
</table>
| uid
name | gid
name |
| ISA
| fromDate |

\[ \langle 142, \text{Bart} \rangle \in \text{User} \langle \text{uid}, \text{name} \rangle \]
\[ \text{Member} \langle \text{uid}, \text{gid}, \text{from}_\text{date} \rangle \]
\[ \langle 456, \text{Ralph}, 😊 \rangle \in \text{PaidUser} \langle \text{uid}, \text{name}, \text{avatar} \rangle \]
```
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

```
<table>
<thead>
<tr>
<th>Users</th>
<th>Groups</th>
</tr>
</thead>
<tbody>
<tr>
<td>uid</td>
<td>gid</td>
</tr>
<tr>
<td>name</td>
<td>name</td>
</tr>
</tbody>
</table>

ISA

<table>
<thead>
<tr>
<th>PaidUsers</th>
<th>IsMemberOf</th>
</tr>
</thead>
<tbody>
<tr>
<td>avatar</td>
<td>fromDate</td>
</tr>
</tbody>
</table>

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

\{142, Bart, NULL\} ∈ Group (gid, name)
\{456, Ralph, 😊\} ∈ User (uid, name, avatar)
Member (uid, gid, from_date)
Comparison of three approaches

• **Entity-in-all-superclasses**
  • *User (uid, name), PaidUser (uid, avatar)*
  • Pro: All users are found in one table
  • Con: Attributes of paid users are scattered in different tables

• **Entity-in-most-specific-class**
  • *User (uid, name), PaidUser (uid, name, avatar)*
  • Pro: All attributes of paid users are found in one table
  • Con: Users are scattered in different tables

• **All-entities-in-one-table**
  • *User (uid, [type, ]name, avatar)*
  • Pro: Everything is in one table
  • Con: Lots of NULL’s; complicated if class hierarchy is complex
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 (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 \((number, \text{engineer, type})\)
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