Relational Database Design: E/R-Relational Translation

Introduction to Databases
CompSci 316 Fall 2019

DUKE COMPUTER SCIENCE
Announcements (Wed. Sep. 4)

• Office hours finalized
  • See website “Help” section

• Gradiance RA exercise due today
  • No late submissions, but we will automatically drop your lowest two scores in the semester

• Gradiance ER exercise assign today; due in a week

• Homework 1 due in 1½ week
  • Please please please start early

• Project description to be posted next week
Announcements (Wed. Sep. 4)

• An *experimental* RA debugger for Homework 1 Problem 1
  • Grew out of research from Zhengjie Miao
  • To be released soon
  • You are not required to use it, but the bonus is that
    • It uses the same (hidden) test db as the autograder
    • If your query is wrong, it will “explain” how, with a very simple example db (with tuples drawn from the hidden test db)
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 $(\text{uid}, \text{name})$  
Group $(\text{gid}, \text{name})$
Translating weak entity sets

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

```
Rooms (building_name, room_number, capacity)
Buildings (name, year)
Seats (building_name, room_number, seat_number, left_or_right)
```

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

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

Parent \((parent\_uid, child\_uid)\)

Member \((uid, initiator\_uid, gid)\)

Key?
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

![Diagram of entity-relationship model]

\[
\begin{align*}
\langle 142, \text{Bart} \rangle & \in \text{User (uid, name)} \\
\langle 456, \text{Ralph} \rangle & \in \text{User (uid, name)} \\
\langle 456, \text{☐} \rangle & \in \text{PaidUser (uid, avatar)} \\
\end{align*}
\]
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

![Database Diagram]

<table>
<thead>
<tr>
<th>Users</th>
<th>IsMemberOf</th>
<th>Groups</th>
</tr>
</thead>
<tbody>
<tr>
<td>gid</td>
<td>fromDate</td>
<td></td>
</tr>
<tr>
<td>name</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>PaidUsers</th>
<th>ISA</th>
<th>Group (gid, name)</th>
</tr>
</thead>
<tbody>
<tr>
<td>avatar</td>
<td></td>
<td>(142, Bart) ∈ User (uid, name)</td>
</tr>
<tr>
<td></td>
<td>ISA</td>
<td>Member (uid, gid, from_date)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>(456, Ralph, 😊) ∈ PaidUser (uid, name, avatar)</td>
</tr>
</tbody>
</table>
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
Comparison of three approaches

• Entity-in-all-superclasses
  • User (uid, name), PaidUser (uid, avatar)
  • Pro:
  • 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

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

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

- **ExpressTrainStop** (express_train_number, time)
  - ExpressTrainStopsAtStation (express_train_number, time, express_station_name)

- **LocalTrainStop** (local_train_number, time)
  - LocalTrainStopsAtStation (local_train_number, time, station_name)

- **Stations**
  - LocalStations
  - ExpressStations

- **time**

- **ISA**

- **merge**
Simplifications and refinements

Train \((\text{number}, \text{engineer})\), LocalTrain \((\text{number})\), ExpressTrain \((\text{number})\)
Station \((\text{name}, \text{address})\), LocalStation \((\text{name})\), ExpressStation \((\text{name})\)
LocalTrainStop \((\text{local_train_number}, \text{station_name}, \text{time})\)
ExpressTrainStop \((\text{express_train_number}, \text{express_station_name}, \text{time})\)

• 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)

• 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

http://ungenius.files.wordpress.com/2010/03/th homer.jpg