Relational Database Design: E/R-Relational Translation

Introduction to Databases
CompSci 316 Fall 2020
Announcements (Tue. Sep. 8)

• HW2 (written + gradiance) due tonight (11:59 pm)
  • No late days for gradiance
  • See late policy for written HWs

• Your team members due tonight
  • Then we will form remaining groups 😊
<table>
<thead>
<tr>
<th>Team</th>
<th>Stadium</th>
<th>Day</th>
</tr>
</thead>
<tbody>
<tr>
<td>T1</td>
<td>S1</td>
<td>D1</td>
</tr>
<tr>
<td>T1</td>
<td>S2</td>
<td>D1</td>
</tr>
<tr>
<td>T2</td>
<td>S3</td>
<td>D2</td>
</tr>
</tbody>
</table>

(A)

<table>
<thead>
<tr>
<th>Team</th>
<th>Stadium</th>
<th>Day</th>
</tr>
</thead>
<tbody>
<tr>
<td>T1</td>
<td>S1</td>
<td>D1</td>
</tr>
<tr>
<td>T2</td>
<td>S2</td>
<td>D2</td>
</tr>
<tr>
<td>T3</td>
<td>S3</td>
<td>D3</td>
</tr>
</tbody>
</table>

(B)

<table>
<thead>
<tr>
<th>Team</th>
<th>Stadium</th>
<th>Day</th>
</tr>
</thead>
<tbody>
<tr>
<td>T1</td>
<td>S1</td>
<td>D1</td>
</tr>
<tr>
<td>T1</td>
<td>S1</td>
<td>D2</td>
</tr>
<tr>
<td>T2</td>
<td>S1</td>
<td>D3</td>
</tr>
</tbody>
</table>

(C)

<table>
<thead>
<tr>
<th>Team</th>
<th>Stadium</th>
<th>Day</th>
</tr>
</thead>
<tbody>
<tr>
<td>T1</td>
<td>S1</td>
<td>D1</td>
</tr>
<tr>
<td>T2</td>
<td>S1</td>
<td>D2</td>
</tr>
<tr>
<td>T3</td>
<td>S1</td>
<td>D1</td>
</tr>
</tbody>
</table>

(D)
Stat...
Quick clarifications: RA questions

• What is “some”?  
  • At least one  
  • e.g., Drinkers frequent some bars that serve beer X  
  • = Drinker D in the answer can frequent bar B1, B2, B3. At least one of them should serve X. It is okay if other bars serve X too.

• What is “only”?  
  • E.g., Drinkers frequent only bars that serve beer X  
  • = If drinker D in the answer frequents a bar B, then B serves X

• What is “every”?  
  • E.g., Drinkers frequent every bars that serve beer X  
  • = If bar B serves beer X, then drinker D in the answer frequents B.
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
You designed an ER digram

Translate it to a Relational Database

Train (number, engineer, type)
Station (name, address, type)
TrainStop (train_number, station_name, time)
E/R model: concepts

• 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
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

How do the keys change if you have arrow to Users, Groups, or both?
More examples

- Users
  - Initiator
  - Member

- Groups
  - Parent
  - Child

- Users
  - IsMemberOf
  - IsParentOf

- Groups
  - Initiator
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
Translating subclasses & ISA
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

```
Users
- uid
- name

Groups
- gid
- name

PaidUsers
- avatar

IsMemberOf

User (uid, 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

```
users
<table>
<thead>
<tr>
<th>uid</th>
<th>name</th>
</tr>
</thead>
</table>

paidusers
<table>
<thead>
<tr>
<th>uid</th>
<th>name</th>
<th>avatar</th>
</tr>
</thead>
</table>

groups
<table>
<thead>
<tr>
<th>gid</th>
<th>name</th>
</tr>
</thead>
</table>

isMemberOf

fromDate

ISA

Group (gid, name)

142, Bart ∈ User (uid, name)

Member (uid, gid, from_date)

456, Ralph, 😊 ∈ PaidUser (uid, name, 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

```
<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>

ISA

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

\[(142, \text{Bart}, \text{NULL}) \]  \[(456, \text{Ralph, :)}) \]

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

• Design a database consistent with the following:
  • A station has a unique name and an address, and is either an express station or a local station
  • A train has a unique number and an engineer, and is either an express train or a local train
  • A local train can stop at any station
  • An express train only stops at express stations
  • A train can stop at a station for any number of times during a day
  • Train schedules are the same everyday

• Draw the ER diagram and translate into relational model
A complete example: ER diagram

Next, Relational Design!

Train no. and time uniquely determine the stop

Why no double diamond?
A complete example: ER diagram

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

• 10 to 8 relations

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 $\pi_{\text{number}}(\text{Station}) - \text{ExpressStation}$

• 8 to 6 relations, still 6 (and more work for queries)!
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 (we will see soon)

• Arguably a better design because it is simpler!
Warning: mechanical translation procedures given in this lecture are no substitute for your own judgment!
What is a “Good” design often depends on your requirements, expected actions, and datasets
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**