Relational Database Design: E/R-Relational Translation

Introduction to Databases
CompSci 316 Fall 2022
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
Case study 1

• Design a database representing cities, counties, and states
  • For states, record name and capital (city)
  • For counties, record name, area, and location (state)
  • For cities, record name, population, and location (county and state)

• Assume the following:
  • Names of states are unique
  • Names of counties are only unique within a state
  • Names of cities are only unique within a county
  • A city is always located in a single county
  • A county is always located in a single state
Case study 1: first design

- County area information is repeated for every city in the county
  - Redundancy is bad (why?)
- State capital should really be a city
  - Should “reference” entities through explicit relationships
Case study 1: second design

- Technically, nothing in this design prevents a city in state $X$ from being the capital of another state $Y$, but oh well...
Case study 2

• 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 every day
Case study 2: first design

- Nothing in this design prevents express trains from stopping at local stations
  - We should capture as many constraints as possible
- A train can stop at a station only once during a day
  - We should not introduce unintended constraints
Is the extra complexity worth it?
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
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
\((\text{room\_building\_name}, \text{room\_number}, \text{building\_name})\)

is subsumed by

Room \((\text{building\_name}, \text{room\_number}, \text{capacity})\)
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)

```plaintext
〈142, Bart〉 ∈ User (uid, name)
〈456, Ralph〉 ∈ User (uid, name)
Member (uid, gid, from_date)
〈456, 😊〉 ∈ 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
  - uid
  - name

Groups
  - gid
  - name

PaidUsers
  - uid
  - name
  - avatar

ISMemberOf

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
Comparison of three approaches

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

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

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

Train \( (\text{number}, \text{engineer}) \)
LocalTrain \( (\text{number}) \)
ExpressTrain \( (\text{number}) \)
Station \( (\text{name}, \text{address}) \)
LocalStation \( (\text{name}) \)
ExpressStation \( (\text{name}) \)

LocalTrainStop \( (\text{local_train_number}, \text{time}) \)
LocalTrainStopsAtStation \( (\text{local_train_number}, \text{time}, \text{station_name}) \)
ExpressTrainStop \( (\text{express_train_number}, \text{time}) \)
ExpressTrainStopsAtStation \( (\text{express_train_number}, \text{time}, \text{express_station_name}) \)
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
    \( \pi_{number}(Train) - ExpressTrain \)
  • Slightly harder to check that local_train_number is indeed a local train number

• Eliminate LocalStation table
  • It can be computed as \( \pi_{number}(Station) - ExpressStation \)
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

• Practice ethical design
  • Your database design influences everything built around it and can have profound effects on real people

http://ungenius.files.wordpress.com/2010/03/thehomer.jpg