Relational Database Design: E/R-Relational Translation

Introduction to Database Management

CS348 Fall 2022
E/R Model

- E/R Concepts
- E/R Schema Design
- Next: Translating E/R to relational schema

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

- An entity set translates directly to a table
  - Attributes $\rightarrow$ columns
  - Key attributes $\rightarrow$ key columns

![Entity relationship diagram]

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

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

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

- If we can deduce the general cardinality constraint (0,1) for a component entity set E, then take the primary key attributes for E
- Otherwise, choose primary key attributes of each component entity

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

\[ \text{Owner (uid, gid, fromDate)} \]

• If we can deduce the general cardinality constraint (0,1) for a component entity set E, then take the primary key attributes for E
• Otherwise, choose primary key attributes of each component entity
More examples

Users

IsParentOf

parent

child

Parent (parent_uid, child_uid)
Translating double diamonds?

• No need to translate because the relationship is implicit in the weak entity set’s translation

```
<table>
<thead>
<tr>
<th>number</th>
<th>capacity</th>
</tr>
</thead>
<tbody>
<tr>
<td>Rooms</td>
<td></td>
</tr>
<tr>
<td>In</td>
<td>Buildings</td>
</tr>
<tr>
<td>number</td>
<td>name</td>
</tr>
<tr>
<td>Seats</td>
<td>year</td>
</tr>
</tbody>
</table>
```

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

```
<table>
<thead>
<tr>
<th>Entity</th>
<th>Attribute</th>
</tr>
</thead>
<tbody>
<tr>
<td>Users</td>
<td>uid, name</td>
</tr>
<tr>
<td>Groups</td>
<td>gid, name</td>
</tr>
<tr>
<td>PaidUsers</td>
<td>avatar</td>
</tr>
</tbody>
</table>

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

![Entity-relationship diagram]

- 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

```
Users (
  uid
  name
)

Groups (
  gid
  name
)

IsMemberOf

PaidUsers

avatar

 ISA

User (uid, name, avatar)
Member (uid, gid, fromDate)

Groups (gid, name)

\{142, Bart, NULL\}
\{456, Ralph, 😊\} ∈
```
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

- **Trains**
  - number
  - engineer
  - ISA: LocalTrains
  - ISA: ExpressTrains

- **LocalTrainStops**
  - time

- **LocalStations**
  - name
  - address
  - ISA: LocalTrainStops
  - ISA: ExpressTrainStops

- **Stations**
  - ISA: LocalTrainStops
  - ISA: ExpressTrainStops

- **ExpressTrains**
  - ISA: LocalTrainStops

- **ExpressStations**
  - ISA: LocalTrainStops
  - ISA: ExpressTrainStops
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 \((number, engineer, type)\)

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

\(\text{Arguably a better design because it is simpler!}\)
Design principles

- **KISS**
  - Keep It Simple, Stupid

- Avoid redundancy

- 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

[POOR DESIGN!](http://ungenus.files.wordpress.com/2010/03/thehomer.jpg)
More examples

• Representing aggregation
  • Tabular representation of aggregation of $R = \text{tabular representation for relationship set } R$
  • To represent relationship set involving aggregation of $R$, treat the aggregation like an entity set whose primary key is the primary key of the table for $R$

$\text{Student (StudentNum) }$
$\text{Couse(CourseNum) }$
$\text{Account(UserID) }$
$\text{EnrolledIn(StudentNum,CourseNum) }$
$\text{CouseAccount(UserID, StudentNum, CourseNum, ExpirationDate) }$

One-to-one relationships $\rightarrow$ We can simply take UserID or (StudentNum, CourseNum) as the key
More examples (Exercise)

- ER Diagram

Relational Schema

?
More examples

• ER Diagram

Relational Diagram
More examples

- ER Diagram

Relational DDL Commands

CREATE TABLE Course
(CourseNum INTEGER PRIMARY KEY,
CourseName CHAR(50));

CREATE TABLE Professor
(ProfNum INTEGER PRIMARY KEY,
ProfName CHAR(50));

CREATE TABLE Section
(CourseNum INTEGER NOT NULL REFERENCES Course(CourseNum),
SectionNum INTEGER NOT NULL,
Term INTEGER NOT NULL,
 PRIMARY KEY(CourseNum, SectionNum, Term),
ProfNum INTEGER NOT NULL REFERENCES Professor(ProfNum));

CREATE TABLE Off-Site Section
(CourseNum INTEGER NOT NULL,
SectionNum INTEGER NOT NULL,
Term INTEGER NOT NULL,
 FOREIGN KEY(CourseNum,SectionNum,Term) REFERENCES Section(CourseNum,SectionNum,Term),
Location CHAR(50));

CREATE TABLE EnrolledIn
(CourseNum INTEGER NOT NULL,
SectionNum INTEGER NOT NULL,
Term INTEGER NOT NULL,
StudentNum INTEGER NOT NULL REFERENCES Student(StudentNum),
 FOREIGN KEY(CourseNum,SectionNum,Term) REFERENCES Section(CourseNum,SectionNum,Term),
Primary Key(CourseNum,SectionNum,Term,StudentNum),
Mark INTEGER);
Database Design

• Entity-Relationship (E/R) model

• Translating E/R to relational schema

• Next week: Relational design principles