Relational Database Design: E/R-Relational Translation

CS348 Spring 2023
Sections: 002 & 004 only
E/R Model

- E/R Concepts
- E/R Schema Design

- Next: Translating E/R to relational schema

![Diagram of E/R Model]

- 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 → 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

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

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

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

Parent \((parent\_uid, child\_uid)\)
Translating double diamonds?

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

Relationship
RoomInBuilding
(room_building_name, room_number,)

is subsumed by entity
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>gid</th>
<th>name</th>
</tr>
</thead>
<tbody>
<tr>
<td>142</td>
<td>Bart</td>
</tr>
<tr>
<td>456</td>
<td>Ralph</td>
</tr>
</tbody>
</table>

Group (gid, name) ∈ User (uid, name)
Member (uid, gid, from_date)
<456, 😊> ∈ PaidUser (uid, avatar)
```
Translating subclasses & ISA: approach 2

- **Entity-in-most-specific-class** approach
  - 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

![Diagram]

Group \((gid, name)\)

\(\langle 142, \text{Bart} \rangle \in \text{User} (uid, name)\)

Member \((uid, gid, from\_date)\)

\(\langle 456, \text{Ralph}, \smile \rangle \in \text{PaidUser} (uid, name, avatar)\)
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

**Example:**

```
Users

uid
name

PaidUsers

avatar

 ISA

IsMemberOf

fromDate

Groups

gid
name

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

(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
Translating composite and multi-valued attributes

**Composite:**
Employee(eId,...,Street, City, Province,..)

**Multi-valued:**
EmployeeHobbies(eID, hobby)
Foreign key: **eId references Employee**
Employee join EmployeeHobbies to get all info
A complete example

Remember Case study 2 exercise?

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
A complete example

- Trains
  - LocalTrainStops
    - time
  - LocalTrains
  - ExpressTrains
    - time
  - ExpressTrainStops
- Stations
  - LocalStations
  - ExpressStations
  - time
  - ISA
- number
  - ISA
- engineer

- name
- address
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 \( \pi_{\text{name}}(\text{Station}) - \text{ExpressStation} \)
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
☞ Arguably a better design because it is simpler!
Design principles

• 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
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 $R$

Student (StudentNum)
Course(CourseNum)
Account(UserID)
EnrolledIn(StudentNum,CourseNum)
CourseAccount(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

[Diagram of ER model showing relationships between Course, CourseName, Section, SectionNum, Term, TaughtBy, EnrolledIn, Off-Site Section, Professor, Student, GPA, CourseNum, CourseName, SectionNum, ProfNum, StudentNum, Location, Mark, StudentNum, GPA, EnrolledIn, CourseNum, CourseName, SectionNum, Term, ProfNum, StudentNum, Term, Mark, ProfNum, ProfName]
More examples

- **ER Diagram**

![ER Diagram Image]

**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 `Student` (StudentNum INTEGER PRIMARY KEY, StudentName CHAR(50), GPA FLOAT);
- 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 lecture (ONLINE): Relational design principles