Relational Database Design
Part II

CPS 196.3
Introduction to Database Systems

Announcement
- Reminder of the new schedule: 12:50pm-2:05pm Mondays and Wednesdays
- Homework #1 will be assigned on Wednesday
  - Two relational algebra problems have been posted on the newsgroup
- Details of the course project and presentation will be available next Monday

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

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 an E/R design to a relational schema

Translating entity sets
- An entity set translates directly to a table
  - Attributes → columns
  - Key attributes → key columns

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 → columns
  - Attributes of the relationship set (if any) → columns
  - Multiplicity of the relationship set determines the key of the table

```
<table>
<thead>
<tr>
<th></th>
<th>Students</th>
<th>Enroll</th>
<th>Course</th>
</tr>
</thead>
<tbody>
<tr>
<td>SID</td>
<td>name</td>
<td></td>
<td>CID</td>
</tr>
<tr>
<td>name</td>
<td></td>
<td>grade</td>
<td>title</td>
</tr>
</tbody>
</table>
```

More examples

```
<table>
<thead>
<tr>
<th></th>
<th>Students</th>
<th>Enroll</th>
<th>Courses</th>
</tr>
</thead>
<tbody>
<tr>
<td>SID</td>
<td>name</td>
<td>TID</td>
<td>name</td>
</tr>
<tr>
<td>name</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
```

Translating double diamonds

- Recall that a double-diamond 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

```
<table>
<thead>
<tr>
<th></th>
<th>Rooms</th>
<th>Buildings</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td></td>
<td>name</td>
</tr>
<tr>
<td>year</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
```

```
RoomInBuilding (room building name, room number, building name)
```

Translating subclasses & ISA (approach 1)

- Entity-in-all-superclasses approach
  - An entity is represented in the table for each subclass to which it belongs
  - A table includes only the attributes attached to the corresponding entity set, plus the inherited key

```
<table>
<thead>
<tr>
<th>SID</th>
<th>name</th>
<th>office</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>GradStudents</td>
</tr>
</tbody>
</table>
```

```
GradStudent (SID, name, office)
```

Translating subclasses & ISA (approach 2)

- Entity-in-most-specific-class approach
  - An entity is only represented in one table (corresponding to 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

```
<table>
<thead>
<tr>
<th>SID</th>
<th>name</th>
<th>office</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>GradStudents</td>
</tr>
</tbody>
</table>
```

```
GradStudent (SID, name, office)
```

Translating subclasses & ISA (approach 3)

- All-entities-in-one-table approach
  - One relation for the root entity set, with all attributes found anywhere in the network of subclasses
  - Use a special NULL value in columns that are not relevant for a particular entity

```
<table>
<thead>
<tr>
<th>SID</th>
<th>name</th>
<th>office</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>GradStudents</td>
</tr>
</tbody>
</table>
```

```
GradStudent (SID, name, office)
```

```
444, "Apu", "D444"
```

```
444, "Apu", "D444"
```

```
444, "Apu", "D444"
```

```
444, "Apu", "D444"
```

```
444, "Apu", "D444"
```

```
444, "Apu", "D444"
```

```
444, "Apu", "D444"
```

```
444, "Apu", "D444"
```

```
444, "Apu", "D444"
```

```
444, "Apu", "D444"
```

```
444, "Apu", "D444"
```

```
444, "Apu", "D444"
```

```
444, "Apu", "D444"
```

```
444, "Apu", "D444"
```

```
444, "Apu", "D444"
```

```
444, "Apu", "D444"
```

```
444, "Apu", "D444"
```

```
444, "Apu", "D444"
```

```
444, "Apu", "D444"
```

```
444, "Apu", "D444"
```

```
444, "Apu", "D444"
```

```
444, "Apu", "D444"
```

```
444, "Apu", "D444"
```

```
444, "Apu", "D444"
```

```
444, "Apu", "D444"
```

```
444, "Apu", "D444"
```

```
444, "Apu", "D444"
```

```
444, "Apu", "D444"
```

```
444, "Apu", "D444"
```

```
444, "Apu", "D444"
```

```
444, "Apu", "D444"
```

```
444, "Apu", "D444"
```

```
444, "Apu", "D444"
```

```
444, "Apu", "D444"
```

```
444, "Apu", "D444"
```

```
444, "Apu", "D444"
```

```
444, "Apu", "D444"
```

```
444, "Apu", "D444"
```

```
444, "Apu", "D444"
```

```
444, "Apu", "D444"
```

```
444, "Apu", "D444"
```

```
444, "Apu", "D444"
```

```
444, "Apu", "D444"
```

```
444, "Apu", "D444"
```

```
444, "Apu", "D444"
```

```
444, "Apu", "D444"
```

```
444, "Apu", "D444"
```

```
444, "Apu", "D444"
```

```
444, "Apu", "D444"
```

```
444, "Apu", "D444"
```

```
444, "Apu", "D444"
```

```
444, "Apu", "D444"
```

```
444, "Apu", "D444"
```

```
444, "Apu", "D444"
```

```
444, "Apu", "D444"
```
Comparison of three approaches

- **Entity-in-all-superclasses**
  - Student (SID, name), GradStudent (SID, office)
  - Pro: All students are found in one table
  - Con: Attributes of grad students are scattered in different tables

- **Entity-in-most-specific-class**
  - Student (SID, name), GradStudent (SID, name, office)
  - Pro: All attributes of grad students are found in one table
  - Con: Students are scattered in different tables

- **All-entities-in-one-table**
  - Student (SID, name), GradStudent (SID, name, office)
  - Pro: Everything is in one table
  - Con: Too many NULL’s; complicated if class hierarchy is complex

A complete example

An alternative design

- Encode the type of train/station as a column rather than creating subclasses
- Difficult to enforce some constraints
  - Type must be either “local” or “express”
  - Express trains only stop at express stations
- Fortunately, they can be expressed/declared explicitly as database constraints in SQL
- Arguably a better design because it is simpler!

Design principles

- **KISS**
  - Keep It Simple, Stupid!
- **Avoid redundancy**
  - Redundancy wastes space, complicates updates, and promotes inconsistency
- **Use your common sense**
  - Warning: Mechanical translation procedures given in this lecture are no substitute for your own judgment