CMSC424: Database Design

Instructor: Amol Deshpande
amol@cs.umd.edu
Today’s Plan

- New topics to discuss
  - More constructs in E/R modeling
  - Converting from E/R to relational schema
  - Creating some E/R models
  - Ruby on Rails

- Other things
  - Grading of SQL: today or tomorrow
  - Assignment/Project 2 released yesterday
    - Make sure the basic setup is working – many more configuration issues compared to PostgreSQL
  - Will start collecting in-class worksheets
Outline

- Entity-relationship Model (E/R model)
- Converting from E/R to Relational
- Ruby on Rails
Recursive Relationships

- Sometimes a relationship associates an entity set to itself
- Need “roles” to distinguish

```
course
  course_id
  title
  credits

prereq
  course_id
  prereq_id
```
Weak Entity Sets

- An entity set without enough attributes to have a primary key
- E.g. Section Entity
- Still need to be able to distinguish between weak entities
  - Called “discriminator attributes”: dashed underline

```
course
  course_id
  title
  credits

sec_course

section
  sec_id
  semester
  year
```
Ternary Relationships

- Sometimes needed, and useful
- Can replace with another entity set and three binary relationships

![E-R Diagram with a ternary relationship.](image)
Specialization/Generalization

Similar to object-oriented programming: allows inheritance etc.

Disjoint vs Overlapping:
No person can be both employee and student

Partial vs Total
There may be “Persons” who are neither employee or student

Different ways to convert to a Relational schema based on the above issues
Aggregation

- No relationships allowed between relationships
- Suppose we want to record evaluations of a student by a guide on a project
Thoughts...

- Nothing about actual data
  - How is it stored?
- No talk about the query languages
  - How do we access the data?
- Semantic vs Syntactic Data Models
  - Remember: E/R Model is used for conceptual modeling
  - Many conceptual models have the same properties
- They are much more about representing the knowledge than about database storage/querying
Basic design principles

- Faithful
  - Must make sense
- Satisfies the application requirements
- Models the requisite domain knowledge
  - If not modeled, lost afterwards
- Avoid redundancy
  - Potential for inconsistencies
- Go for simplicity

Typically an iterative process that goes back and forth
Entity sets vs attributes
- Depends on the semantics of the application
- Consider *telephone*

![Diagram showing entity sets and relationships](image-url)

**Diagram**

(a) `instructor`:
- ID
- name
- salary
- phone_number

(b) `instructor`
- ID
- name
- salary

`inst_phone`:
- Phone
- phone_number
- location

**Figure 7.17** Alternatives for adding `phone` to the `instructor` entity set.
Design Issues

- Entity sets vs Relationship sets
  - Consider *takes*

Figure 7.18  Replacement of *takes* by *registration* and two relationship sets
Design Issues

- Entity sets vs attributes
  - Depends on the semantics of the application
  - Consider telephone

- Entity sets vs Relationship sets
  - Consider takes

- N-ary vs binary relationships
  - Possible to avoid n-ary relationships, but there are some cases where it is advantageous to use them

- It is not an exact science !!
Recap

- Entity-relationship Model
  - Intuitive diagram-based representation of domain knowledge, data properties etc...
  - Two key concepts:
    - Entities
    - Relationships
  - We also looked at:
    - Relationship cardinalities
    - Keys
    - Weak entity sets
    - ...

Entity-relationship Model

- No standardized model (as far as I know)
  - You will see different types of symbols/constructs

- Easy to reason about/understand/construct

- Not as easy to implement
  - Came after the relational model, so no real implementation was ever done
  - Mainly used in the design phase
Outline

- Entity-relationship Model (E/R model)
- Converting from E/R to Relational
- Ruby on Rails
E/R Diagrams → Relations

- Convert entity sets into a relational schema with the same set of attributes

Customer

- cname
- ccity
- cstreet

Customer_Schema(cname, ccity, cstreet)

Branch

- bname
- bcity
- assets

Branch_Schema(bname, bcity, assets)
Convert relationship sets *also* into a relational schema

Remember: A relationship is completely described by primary keys of associate entities and its own attributes

Well… Not quite. We can do better. It depends on the relationship cardinality
Say One-to-Many Relationship from Customer to Account
→ Many accounts per customer

Account_Schema(acct-no, balance, cname, access-date)

Customer_Schema(cname, ccity, cstreet)
## E/R Diagrams → Relations

<table>
<thead>
<tr>
<th>E/R</th>
<th>Relational Schema</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Entity Sets</strong></td>
<td>E = (a₁, …, aₙ)</td>
</tr>
<tr>
<td>E₁</td>
<td></td>
</tr>
<tr>
<td>a₁ … aₙ</td>
<td></td>
</tr>
<tr>
<td><strong>Relationship Sets</strong></td>
<td>R = (a₁, b₁, c₁, …, cₙ)</td>
</tr>
<tr>
<td>E₁</td>
<td></td>
</tr>
<tr>
<td>a₁ … aₙ</td>
<td></td>
</tr>
<tr>
<td>E₂</td>
<td></td>
</tr>
<tr>
<td>b₁ … bₘ</td>
<td></td>
</tr>
<tr>
<td>R</td>
<td>c₁ … cₖ</td>
</tr>
</tbody>
</table>

*Not the whole story for Relationship Sets ...*
## E/R Diagrams \( \rightarrow \) Relations

<table>
<thead>
<tr>
<th>Relationship Cardinality</th>
<th>Relational Schema</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>n:m</strong></td>
<td>( E_1 = (a_1, \ldots, a_n) )</td>
</tr>
<tr>
<td></td>
<td>( E_2 = (b_1, \ldots, b_m) )</td>
</tr>
<tr>
<td></td>
<td>( R = (a_1, b_1, c_1, \ldots, c_n) )</td>
</tr>
<tr>
<td><strong>n:1</strong></td>
<td>( E_1 = (a_1, \ldots, a_n, b_1, c_1, \ldots, c_n) )</td>
</tr>
<tr>
<td></td>
<td>( E_2 = (b_1, \ldots, b_m) )</td>
</tr>
<tr>
<td><strong>1:n</strong></td>
<td>( E_1 = (a_1, \ldots, a_n) )</td>
</tr>
<tr>
<td></td>
<td>( E_2 = (b_1, \ldots, b_m, a_1, c_1, \ldots, c_n) )</td>
</tr>
<tr>
<td><strong>1:1</strong></td>
<td>Treat as n:1 or 1:n</td>
</tr>
</tbody>
</table>

### Diagram

- \( E_1 \) and \( E_2 \) are entities.
- \( R \) is the relationship.
- \( a_1, \ldots, a_n \) and \( b_1, \ldots, b_m \) are attributes.
- \( c_1, \ldots, c_k \) are additional attributes in the relationship set. 

Treat as n:1 or 1:n relationships.
Q. How many tables does this get translated into?

A. 6 (account, branch, customer, loan, depositor, borrower)
**E/R Diagrams & Relations**

<table>
<thead>
<tr>
<th>E/R</th>
<th>Relational Schema</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Weak Entity Sets</strong></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>E₁</th>
<th>E₂</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="image1" alt="E₁ Diagram" /></td>
<td><img src="image2" alt="E₂ Diagram" /></td>
</tr>
</tbody>
</table>

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>E₁ = (a₁, ..., aₙ)</td>
<td>E₂ = (a₁, b₁, ..., bₘ)</td>
</tr>
</tbody>
</table>
### E/R Diagrams & Relations

#### E/R

<table>
<thead>
<tr>
<th>Employee</th>
<th>ssn</th>
<th>name</th>
<th>phone</th>
</tr>
</thead>
</table>

#### Relational Schema

- **Emp** = (ssn, name)
- **Emp-Phones** = (ssn, phone)

<table>
<thead>
<tr>
<th>ssn</th>
<th>name</th>
<th>phone</th>
</tr>
</thead>
<tbody>
<tr>
<td>001</td>
<td>Smith</td>
<td>4-1234</td>
</tr>
<tr>
<td></td>
<td></td>
<td>4-5678</td>
</tr>
</tbody>
</table>

Emp

Emp-Phones
### E/R Diagrams & Relations

<table>
<thead>
<tr>
<th>E/R</th>
<th>Relational Schema</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Subclasses</strong></td>
<td></td>
</tr>
</tbody>
</table>

![E/R Diagram](image)

**Method 1:**

- \( E = (a_1, \ldots, a_n) \)
- \( E_1 = (a_1, b_1, \ldots, b_m) \)
- \( E_2 = (a_1, c_1, \ldots, c_k) \)

**Method 2:**

- \( E_1 = (a_1, \ldots, a_n, b_1, \ldots, b_m) \)
- \( E_2 = (a_1, \ldots, a_n, c_1, \ldots, c_k) \)
Subclasses example:

Method 1:

- Account = (acct_no, balance)
- SAccount = (acct_no, interest)
- CAccount = (acct_no, overdraft)

Method 2:

- SAccount = (acct_no, balance, interest)
- CAccount = (acct_no, balance, overdraft)
Outline

- Entity-relationship Model (E/R model)
- Converting from E/R to Relational
- Ruby on Rails
Ruby on Rails: Overview

- Web application framework written in Ruby
- Uses a **Model-View-Controller** pattern
- Paradigms:
  - Convention-over-configuration
  - Don’t repeat yourself
  - Active record pattern
Some Details

- **Models (stored in app/models)**
  - Map to database tables
  - E.g., Instructor model maps to instructors table in the database
  - Can keep track of associations between different models (e.g., relationships, foreign keys, etc).

- **Controllers (stored in app/controllers)**
  - Responds to external requests from the web server
  - Manipulates the models and passes information to views

- **Views (stored in app/views)**
  - ERB files that control what is shown to the user
  - Get translated into HTML

- **Config/routes.rb**
  - Dictates what controllers to call in response to different requests
Project

- Basic skeleton already created for you
- You have to figure out what more entities to add, and set up the controllers/views/models appropriately
  - Probably don’t need to add anything to config/routes.rb

- To get started:
  - Create the student listing and course listing pages
  - Understand how to set up associations between different models (see appropriate Rails guide)
More Information

- Many tutorials on the web
- Rails Guides are very good
- Rails for Zombies great place to get started