# ER & Relational: Digging Deeper

R &G - Chapters 2 & 3

*Agile Web Development with Rails 3rd edition*
Chapters 18-19.3

## Databases Model the Real World

- Data Model: "language" of concepts to translate real world into data
- Many models: Relational, E-R, O-O, Network, Hierarchical, etc.
- Relational
  - Rows & Columns
  - Keys & Foreign Keys to link Relations

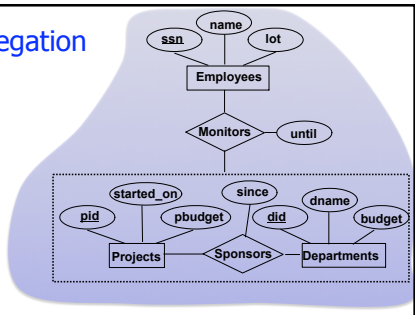
Enrolled

| sid | cid | grade |
|---|---|---|
| 53666 | Carnatic101 | C |
| 53666 | Reggae203 | B |
| 53650 | Topology112 | A |
| 53666 | History105 | B |

Students

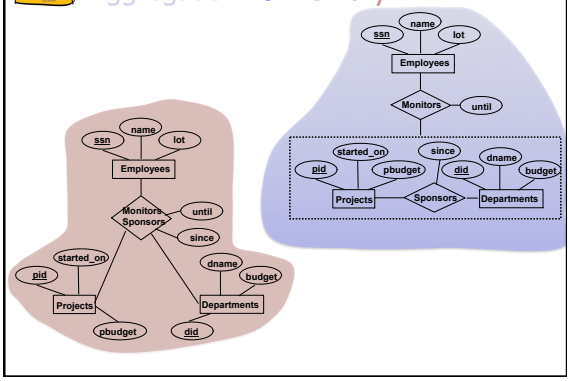
| sid | name | login | age | gpa |
|---|---|---|---|---|
| 53666 | Jones | jones@cs | 18 | 3.4 |
| 53688 | Smith | smith@eecs | 18 | 3.2 |
| 53650 | Smith | smith@math | 19 | 3.8 |

## Aggregation




Allows relationships with *relationship sets*.

## Aggregation vs. Ternary




## Conceptual Design Using the ER Model

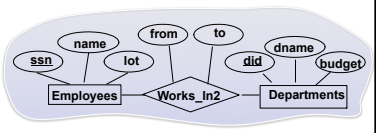
- ER modeling *can* get tricky!
- Design choices:
  - Entity or attribute?
  - Entity or relationship?
  - Relationships: Binary or ternary? Aggregation?
- ER Model goals and limitations:
  - Lots of semantics can (and should) be captured.
  - Some constraints *cannot* be captured in ER.
    - We'll refine things in our logical (relational) design

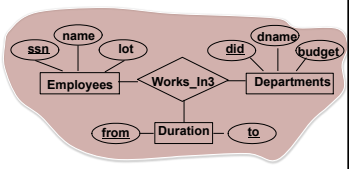
## Entity vs. Attribute

- "Address":
  - attribute of Employees?
  - Entity of its own?
- It depends! Semantics and usage.
  - Several addresses per employee?
    - must be an entity
    - atomic attribute types (no set-valued attributes!)
  - Care about structure? (city, street, etc.)
    - must be an entity!
    - atomic attribute types (no tuple-valued attributes!)

## Entity vs. Attribute (Cont.)

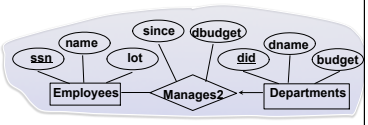
- Works_In2: employee cannot work in a department for >1 period.
- Like multiple addresses per employee!

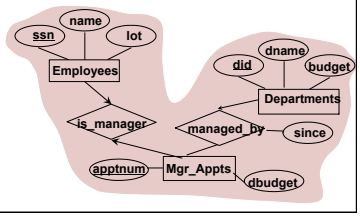



## Entity vs. Relationship

- Separate discretionary budget (dbudget) for each dept.
- What if manager's dbudget covers all managed depts
  - Could repeat value
  - But redundancy = problems
- Better design:



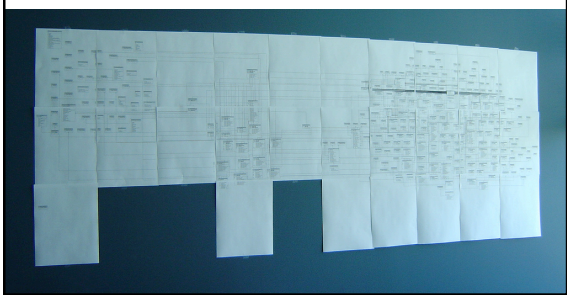

## Now you try it

Try this at home - Courses database:

- Courses, Students, Teachers
- Courses have ids, titles, credits, …
- Courses have multiple sections that have time/rm and exactly one teacher
- Must track students' course schedules and transcripts including grades, semester taken, etc.
- Must track which classes a professor has taught
- Database should work over multiple semesters

## E-R Diagram as Wallpaper

- Very common for them to be wall-sized

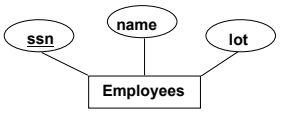


## Converting ER to Relational

- Fairly analogous structure
- But many simple concepts in ER are subtle to specify in relations

## Logical DB Design: ER to Relational

- Entity sets to tables.




| ssn | name | lot |
|---|---|---|
| 123-22-3666 | Attishoo | 48 |
| 231-31-5368 | Smiley | 22 |
| 131-24-3650 | Smethurst | 35 |

```
CREATE TABLE Employees
  (ssn CHAR(11),
   name CHAR(20),
   lot  INTEGER,
   PRIMARY KEY  (ssn))
```

## Relationship Sets to Tables

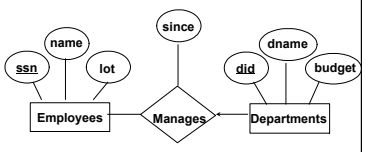
- In translating a many-to-many relationship set to a relation, attributes of the relation must include:
  1) Keys for each participating entity set (as foreign keys). This set of attributes forms a *superkey* for the relation.
  2) All descriptive attributes.

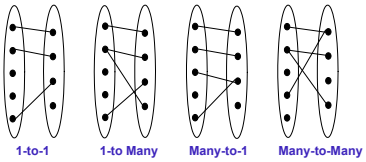
```
CREATE TABLE Works_In(
  ssn  CHAR(1),
  did  INTEGER,
  since  DATE,
PRIMARY KEY (ssn, did),
FOREIGN KEY (ssn)
   REFERENCES Employees,
FOREIGN KEY (did)
 REFERENCES Departments)
```

| ssn | did | since |
|---|---|---|
| 123-22-3666 | 51 | 1/1/91 |
| 123-22-3666 | 56 | 3/3/93 |
| 231-31-5368 | 51 | 2/2/92 |

## Review: Key Constraints

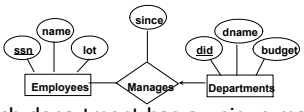
- Each dept has at most one manager, according to the *key constraint* on Manages.




*Translation to relational model?*

## Translating ER with Key Constraints




- Since each department has a unique manager, we could instead combine Manages and Departments.

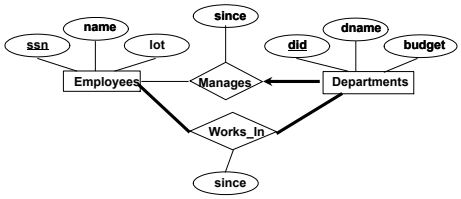
```
CREATE TABLE  Manages(
 ssn  CHAR(11),
 did  INTEGER,
 since  DATE,
 PRIMARY KEY  (did),
 FOREIGN KEY (ssn)
REFERENCES Employees,
   FOREIGN KEY (did)
REFERENCES Departments)
```

Vs.

```
CREATE TABLE  Dept_Mgr(
 did  INTEGER,
 dname  CHAR(20),
 budget  REAL,
 ssn  CHAR(11),
 since  DATE,
 PRIMARY KEY  (did),
 FOREIGN KEY (ssn)
  REFERENCES Employees)
```

## Review: Participation Constraints

- Does every department have a manager?
  - If so, this is a *participation constraint*: the participation of Departments in Manages is said to be *total* (vs. *partial*).
    - Every *did* value in Departments table must appear in a row of the Manages table (with a non-null *ssn* value!)



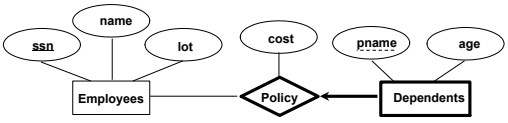

## Participation Constraints in SQL

- We can capture participation constraints involving one entity set in a binary relationship, but little else (without resorting to CHECK constraints which we'll learn later).

```
CREATE TABLE  Dept_Mgr(
   did  INTEGER,
   dname  CHAR(20),
   budget  REAL,
   ssn  CHAR(11) NOT NULL,
   since  DATE,
   PRIMARY KEY  (did),
   FOREIGN KEY  (ssn) REFERENCES
Employees,
      ON DELETE NO ACTION)
```

## Review: Weak Entities

- A *weak entity* can be identified uniquely only by considering the primary key of another (*owner*) entity.
  - Owner entity set and weak entity set must participate in a one-to-many relationship set (1 owner, many weak entities).
  - Weak entity set must have total participation in this *identifying* relationship set.



## Translating Weak Entity Sets

- Weak entity set and identifying relationship set are translated into a single table.
  - When the owner entity is deleted, all owned weak entities must also be deleted.

```
CREATE TABLE  Dep_Policy (
   pname  CHAR(20),
   age  INTEGER,
   cost  REAL,
   ssn  CHAR(11) NOT NULL,
   PRIMARY KEY  (pname, ssn),
   FOREIGN KEY  (ssn) REFERENCES Employees,
      ON DELETE CASCADE)
```

## Administrivia

- Homework #1
- Class website
  - Trac & svn
  - Calendar
  - Syllabus
  - Roadmap
  - Screencasts
  - Grading

## Databases for Programmers

- Programmers think about objects (structs)
  - Nested and interleaved
- Often want to "persist" these things
- Options
  - encode opaquely and store
  - translate to a structured form
    - relational DB, XML file
  - pros and cons?

## Remember the Inequality!

$$\frac{dapp}{dt} << \frac{denv}{dt}$$

- If storing indefinitely…use a flexible representation

## But YUCK!!

- How do I "relationalize" my objects?
- Have to write a converter for each class?
- Think about when to save things into the DB?

- Good news:
  - Can all be automated
  - With varying amounts of trouble

## Object-Relational Mappings

- Roughly:
  - Class ~ Entity Set
  - Instance ~ Entity
  - Data member ~ Attribute
  - Reference ~ Foreign Key

## Details, details

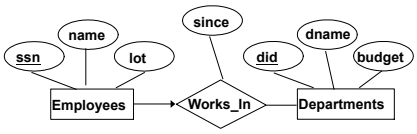
- We have to map this down to tables
- Which table holds which class of object?
- What about relationships?
- Solution #1: Declarative Configuration
  - Write a description file (often in XML)
    - E.g. Enterprise Java Beans (EJBs)
- Solution #2: Convention
  - Agree to use some conventions
    - E.g. Rails

## Ruby on Rails

- Ruby: an OO scripting language
  - and a pretty nice one, too
- Rails: a framework for web apps
  - "convention over configuration"
    - great for standard web-app stuff!
  - allows overriding as needed
- Rails "Models"
  - Represent the data and business rules in an application
  - Very ER-like

## Rails and ER

- Models
  - Employees
  - Departments
  - Works_In?
    - Depends on constraints




## Rails "Models" and "Associations"

*app/models/department.rb*

```
class Department < ActiveRecord::Base
  has_many :employees # 1-to-n
end
```

*app/models/employee.rb*

```
class Employee < ActiveRecord::Base
  belongs_to :department # n-to-1
end
```

## Rails "Models" and "Associations"

*app/models/engine.rb*

```
class Engine < ActiveRecord::Base
  belongs_to :vehicle # 1-to-0 or 1-to-1
end
```

app/models/vehicle.rb

```
Class Vehicle < ActiveRecord::Base
  has_one :engine,   # 1-to-1
          :conditions => "id is not null"
end
```

## Rails "Models" and "Associations"

*app/models/parent.rb*

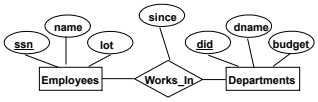
```
Class Parent < ActiveRecord::Base
  # many-to-many
  has_and_belongs_to_many :children
end
```

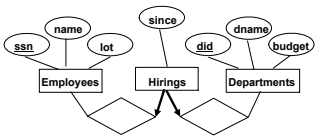
app/models/child.rb

```
Class Child < ActiveRecord::Base
  # many-to-many, full participation
  has_and_belongs_to_many :parents,
          :conditions => "id is not null"
end
```

## A more complex example

- Models
  - Employees
  - Departments
  - Works_In?
    - Remove the key constraint
    - Where does "since" live?? Needs a model!




## Rails "Through Associations"

```
app/models/hiring.rb
Class Hiring < ActiveRecord::Base
  belongs_to :employee, # one-to-many
    :conditions => "id is not null"
  belongs_to :department, # one-to-many
    :conditions => "id is not null"
end

app/models/employee.rb
Class Employee < ActiveRecord::Base
  has_many :hirings
  has_many :departments,
           :through => hirings
end
```

## Further Reading

- Chapter 18/19 (through 19.3) in *Agile Web Development with Rails (3rd edition)*

## Summary of Conceptual Design

- *Conceptual design* follows *requirements analysis*,
  - Yields a high-level description of data to be stored
- ER model popular for conceptual design
  - Constructs are expressive, close to the way people think about their applications.
  - Note: There are many variations on ER model
    - Both graphically and conceptually
- Basic constructs: *entities, relationships*, and *attributes* (of entities and relationships).
- Some additional constructs: *weak entities, ISA hierarchies* (see text if you're curious), and *aggregation*.

## Summary of ER (Cont.)

- Several kinds of integrity constraints:
  - *key constraints*
  - *participation constraints*
- Some *foreign key constraints* are also implicit in the definition of a relationship set.
- Many other constraints (notably, *functional dependencies*) cannot be expressed.
- Constraints play an important role in determining the best database design for an enterprise.

## Summary of ER (Cont.)

- ER design is *subjective*. There are often many ways to model a given scenario!
- Analyzing alternatives can be tricky, especially for a large enterprise. Common choices include:
  - Entity vs. attribute, entity vs. relationship, binary or n-ary relationship, whether or not to use ISA hierarchies, aggregation.
- Ensuring good database design: resulting relational schema should be analyzed and refined further.
  - Functional Dependency information and normalization techniques are especially useful.