Framework for E/R

- **Design is serious business**
  - The boss (or customer) wants a database
  - but has no clue what to put in it

- **Sketching the key components is an efficient way to develop a working database**
  - Sketch out (and debug) schema designs
  - Express as many constraints as possible
  - Convert to relational DB once the boss is happy

The entity/relationship model

**Introduction to databases**
CSCC43 Winter 2011
Ryan Johnson

Thanks to Arnold Rosenbloom and Renee Miller
for material in these slides

**Entity/Relationship model**

- **Visual data model (diagram-based)**
  - Quickly “chart out” a database design
  - Easier to “see” big picture

- **Basic concept:** “things” (entities) and their *relationships* with other entities, along with the *attributes* describing them

- **Framework for E/R**

  - Design is serious business
  - The boss (or customer) wants a database
    - but has no clue what to put in it
  - Sketching the key components is an efficient way to develop a working database
    - Sketch out (and debug) schema designs
    - Express as many constraints as possible
    - Convert to relational DB once the boss is happy

**Entity/Relationship vs other models**

<table>
<thead>
<tr>
<th>E/R</th>
<th>OO</th>
<th>RA</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>“thing” to be modeled</strong></td>
<td><strong>Entity</strong></td>
<td><strong>Object</strong></td>
</tr>
<tr>
<td>set of similar “things”</td>
<td><strong>Entity set</strong></td>
<td><strong>Class</strong></td>
</tr>
<tr>
<td>relationship</td>
<td><strong>Relationship</strong></td>
<td>Object?</td>
</tr>
<tr>
<td>set of similar relationships</td>
<td><strong>Relationship Set</strong></td>
<td>Class?</td>
</tr>
<tr>
<td>property of a “thing” or of a relationship</td>
<td><strong>Attribute</strong></td>
<td><strong>Field</strong></td>
</tr>
</tbody>
</table>
Entity set and relationship set

- Describe classes of entities/relations
  - Entities (and relationships between them) which are allowed to exist
  - Defined using domain knowledge
- Think of both as tables
  - Both can have attributes
- Entity set
  - One column for each attribute
  - One row for each entity
- Relationship set
  - One column for each entity set in the relationship
  - One additional column for each attribute

Multiplicity of relationships

- One-one
- Many-one
- Many-many

An entity might not participate in any relationship

Example of multiplicity

Multiway relationships

- Binary relationships not always enough
  - Actor playing multiple roles?
  - Role requiring several actors?

Can we add any arrowheads?
**Relationships linking one entity set**

- **This is why we need self-joins**
  - OK for one entity to participate in several kinds of relationship

  ![Diagram of relationships](image)

- **Symmetric roles also need edge labels in practice**
  - Edge labels distinguish asymmetric roles

**Inheritance in E/R**

- **Subclass = special case**
  - Fewer instances, more attributes (usually)
  - One-one relationship between classes
  - Attributes: union of classes involved

![Inheritance diagram](image)

- **Roles**
  - **MildMannered**
  - **AlterEgo**
  - **Superheroes**

**Multiple inheritance in E/R**

- **Allowed, but not usually necessary**
  - Entity can “be” many classes (union)

- **Usually not a good idea**
  - Naming collisions, semantic clashes
  - What if both have attribute ‘nominated’?
  - Queries often work just as well

- **Usable classes usually form a tree**

![Multiple inheritance diagram](image)

**E/R != OO**

- **Tempting to favor OO design principles**
  - Usually allowed, but limiting
  - Multi-way relationship (already seen)
  - Heterogenous “hierarchies” (below)

![E/R vs. OO diagram](image)

- **Dogs owned by people**
- **Some actors are dogs**
- **All directors are people**
Keys

• A key is a set of attributes which uniquely identifies every entity in the entity set
  => SIN (people), VIN (car), address (house), ...
  – Two entities can agree on some, but not all, of the key attributes.

• Every entity set requires one key
  – Even if it involves every attribute
  – Choose one if there are multiple candidate keys

• Underline the key attribute(s)

What about duplicates?

Weak entity sets

• Occasionally, entities of an entity set need “help” to identify them uniquely.

• Entity set $E$ is said to be weak if in order to identify entities of $E$ uniquely, we need to follow one or more many-one relationships from $E$ and include the key of the related entities from the connected entity sets.

• Weak entities never exist alone
  – Always at least one supporting relationship to identify them
  – Other relationships allowed as well

Weak entity sets – example

• name is almost a key for football players, but there might be two with the same name

• number is certainly not a key, since players on two teams could have the same number

• But number, together with the team name related to the player by PlaysFor should be unique

Weak entity sets – E/R diagrams

• Double rectangle for the weak entity set
• Double diamond for supporting many-one relationship

Note: must be rounded because every player needs a team to construct a key
“Chained” weak entity sets

Key: host + subdomain + domain

Weak entity sets in practice

• Question: how does a supporting relationship identify an entity having an incomplete key?
  – Example: print servers
    => CS: inkblot, treekiller
    => Math: papershredder, treekiller
  – Answer: it doesn’t.
    – Option 1: replicate Printers (CSPrinters and MathPrinters)
    – Option 2: create/use some artificial key (serial number, etc.)
    – Option 3: store full key in weak entity (most common)

• Weak entities: a (sometimes useful) myth

Recommendation: need a good reason to use

From E/R Diagrams to Relations

• Both entity sets and relationships -> relations
• Attributes for entity set migrate directly
• Attributes for relationships
  – Keys of the connected entity sets
  – Any attributes of the relationship itself

Combining relations

• OK to merge into entity set E’s relation the relations for binary many-one relationships of which E is the “many”

Movies(title, year)
DirectedBy(title, name)

vs.

Movies(title, year, directedBy)
Directors(name, address)
Combining many-many relationships

- Neither E.S. “owns” the other
  - No clear home for attributes
  - Result: redundancy
- Don’t merge them!
  - Inelegant and messy
  - Waste of space
  - Error-prone updates

<table>
<thead>
<tr>
<th>name</th>
<th>address</th>
<th>role</th>
<th>title</th>
<th>year</th>
</tr>
</thead>
<tbody>
<tr>
<td>James Jones</td>
<td>Villa, CA</td>
<td>Vader</td>
<td>Star Wars IV</td>
<td>1977</td>
</tr>
<tr>
<td>James Jones</td>
<td>Villa, CA</td>
<td>Vader</td>
<td>Star Wars V</td>
<td>1980</td>
</tr>
<tr>
<td>James Jones</td>
<td>Villa, CA</td>
<td>Vader</td>
<td>Star Wars VI</td>
<td>1983</td>
</tr>
<tr>
<td>James Jones</td>
<td>Villa, CA</td>
<td>Greer</td>
<td>Red October</td>
<td>1990</td>
</tr>
<tr>
<td>James Jones</td>
<td>Villa, CA</td>
<td>Mustafa</td>
<td>Lion King</td>
<td>1994</td>
</tr>
</tbody>
</table>

Handling Weak Entity Sets

- Relation for a weak entity set must include attributes for its complete key (including those belonging to other entity sets), as well as its own, nonkey attributes.
- A supporting relationship is redundant and yields no relation (unless it has attributes).

Example: weak entity set -> relation

Players(name, number)
Teams(name, city)
PlaysFor(number, teamName, name)

Must know team to identify a player uniquely

PlaysFor is redundant
always equal

Challenge: modeling the “real world”

- Life is arbitrarily complex
  - Directors who are also actors?
  - Actors who play multiple roles in one movie?
  - Roles which require multiple actors?
  - Stunt-doubles? Voice-overs? Cameos?
  - Somebody quits/dies during filming?
  - Animal actors?
- Key to successful model: parsimony
  - As complex as necessary, but no more
  - Choose to represent only “relevant” things

What to do with things model doesn’t capture?
Design Techniques

1. Avoid redundancy
2. Limit the use of weak entity sets
3. Don’t use an entity set if an attribute will do
4. Keep keys simple

Avoiding Redundancy

• **Redundancy** = saying the same thing in two (or more) different ways.
• Wastes space and (more importantly) encourages inconsistency.
  – Two representations of the same fact become inconsistent if we change one and forget to change the other.
• Usually indicates a design flaw as well
  – Example: storing actor’s address with movies
    => Address at time of filming? Now? Hotel near studio?

Two types of redundancy

- Repeated information
- Repeated designs (same or similar attributes)

<table>
<thead>
<tr>
<th>name</th>
<th>address</th>
<th>role</th>
</tr>
</thead>
<tbody>
<tr>
<td>James Jones</td>
<td>Villa, CA</td>
<td>Vader</td>
</tr>
<tr>
<td>James Jones</td>
<td>Villa, CA</td>
<td>Vader</td>
</tr>
<tr>
<td>James Jones</td>
<td>Villa, CA</td>
<td>Vader</td>
</tr>
<tr>
<td>James Jones</td>
<td>Villa, CA</td>
<td>Greer</td>
</tr>
<tr>
<td>James Jones</td>
<td>Villa, CA</td>
<td>Mustafa</td>
</tr>
</tbody>
</table>

Entity Sets Versus Attributes

- An entity set should satisfy at least one of the following conditions:
  – It is more than the name of something; it has at least one nonkey attribute.
    or
  – It is the “many” in a many-one or many-many relationship.
- Rules of thumb
  – A “thing” in its own right => Entity Set
  – A “detail” about some other “thing” => Attribute
  – A “detail” correlated among many “things” => Entity Set

Really this is just about avoiding redundancy
E.S. vs. attributes: bad examples

- Many movies, one role?
- Many roles, one movie?
- Unnecessary entity set and relationship
- Studio has non-key attributes
- Many actors at each studio

When to use weak entity sets?

- The usual reason is that there is no global authority capable of creating unique ID’s.
- Example: it is unlikely that there could be an agreement to assign unique player numbers across all football teams in the world.

Don’t Overuse Weak Entity Sets

- Beginning database designers often doubt that anything could be a key by itself
  - They make all entity sets weak, supported by all other entity sets to which they are linked.
- In reality, each entity gets a unique ID anyway
  - Social insurance number, automobile VIN, etc.
  - Useful for many reasons (next slide)

Keeping keys simple

- Multi-attribute and/or string keys...
- ... are redundant
  - e.g. Movies(title, year, ...): 2 attributes, ~16 bytes
  - Number of movies ever made \(< 2^{32} \) (4 bytes)
  - Integer movieID key saves 75% space and a lot of typing
- ... break encapsulation
  - e.g. Patient(firstName, lastName, phone, ...)
  - Security/privacy hole
  - Integer patientID prevents information leaks
- ... are brittle (nasty interaction of above two points)
  - Name or phone number change? Parent and child with same name?
  - Patient with no phone? Two movies with same title and year?
  - Integer ID always exists, immutable, unique

Also: computers are really good at integers...
Converting N-ary relationship to binary

- Usually not a good idea
  - Increase in complexity, no additional expressiveness
  - But needed for some (object-oriented) data models

Removing attributes from relationships

- Attributes can be moved to their own E.S.

Note: Movie and actor together uniquely identify a salary

Again, usually not a good idea