Today

- Quiz 8
- ER modeling

Assignments

- HW9 due Tues
- Proj 2 due Tues
- Exam 2 in one week
How do these differ?

Q: Which of these is correct?

• Based on the application requirements!

Constraints are expressed over Entity and Relationship Sets

• Constrain the members of the corresponding sets

Q: Does this satisfy the cardinality constraints?
Q: What about now?

Q: And now?
Continuing with ER modeling

There are different notations for writing cardinality constraints ...

- Examples of “one to many” constraints

<table>
<thead>
<tr>
<th>one</th>
<th>many</th>
<th>zero..one</th>
<th>one..many</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>n</td>
<td>0:1</td>
<td>1:n</td>
</tr>
<tr>
<td>1</td>
<td>*</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>0-1</td>
<td>1+</td>
</tr>
</tbody>
</table>

Maximum cardinalities only   Minimum and maximum cardinalities

- Examples of “many to many” constraints

<table>
<thead>
<tr>
<th>many</th>
<th>many</th>
<th>one..many</th>
<th>one..many</th>
</tr>
</thead>
<tbody>
<tr>
<td>m</td>
<td>n</td>
<td>1:n</td>
<td>1:n</td>
</tr>
<tr>
<td>*</td>
<td>*</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>1+</td>
<td>1+</td>
</tr>
</tbody>
</table>

Maximum cardinalities only   Minimum and maximum cardinalities
Relationship Attributes

Similar to Entities, Relationships can have attributes

- Each instance of the relationship set has a value for the attribute

Try all three locations ... where does the attribute make sense?

- Because Employees have zero or one home department
  - start date will work as an Employee or home attribute

- Not clear what it would mean at Department

Q: What if Employee could have multiple home departments?
  - Instead of 0..1, we’d have 0..*
  - In this case, it would need to be a relationship attribute
Role names

Relationships can have role names

- An employee “manages” zero or one department
- A department is “managed by” exactly one employee

Role names help in more complex relationships

- Here Employee participates in different roles for the same relationship set
Weak Entity Sets

In this model assume we ...
- need to record the insurance policies of employees
- need to track dependencies of policies
- only need to store the name and date-of-birth of dependents
- no longer track their policies or dependents when an employee leaves

We also assume dependents identified through their employees
- we assume dependents of an employee have unique names
- different employees could have dependents with the same name

In this case ...
- Employee is the “strong“ entity set
- Policy is the “identifying“ relationship set
- Dependent is the “weak“ entity set
  - it wouldn’t be in the DB if the strong entity were not present
- Dependent name is a “partial“ key
  - must be combined with the strong entity key to identify the dependent
Q: What must the cardinalities be for an identifying relationship set?

- we denote the identifying relationship with double lines
- the weak entity with double lines
- the partial key with a dashed underline
These two schemas are not equivalent!

- The **ternary** relationship implies that:
  
  *a supplier must be authorized to supply a part to a project*

- For example:
  
  - *office depot supplies pencils to project 112*
  - *staples supplies paper to project 115*
  - *which does not* imply *staples supplies pencils to 112*
  - *(but it would in the binary version ...)*
The binary relationships each represent something **distinct**

- For example, let’s say that:
  - both staples and office depot provide pencils
  - office depot also provides paper
  - project 112 requires pencils
  - and project 112 authorizes both office depot and staples (as suppliers)

Q: Who supplies pencils to project 112?

  - Unclear in the binary version ...
  - providing, authorizing, and requiring to not imply supplying

Q: But is it possible to use binary relationships to capture supplying?

  - Yes!
  - Make supplies an entity set with three relationship sets
Dualities: Entities versus Attributes

Q: When should something be modeled as an entity instead of an attribute?
   - If the attribute has attributes or other relationships
   - Attributes are for “simple” (atomic) values
   - Note: ER models do not have foreign keys!

Q: Should office be an attribute or an entity?
   - It (again) depends on the application requirements!
   - Many attributes can be “promoted” to an entity set
   - Some entities can be “demoted” to an attribute value

Q: When would it make sense to model offices as entities?
   - Employees can have more than one office
   - Additional attributes (like area, location, etc.) of offices
   - Offices are used in other relationships
     * e.g., to store the furniture, telephones, network drops in the offices

This is an example of why modeling can be hard ...

- and why it is important to understand the requirements
Translating ERDs to Relational Schemas

Entity sets
- Each entity becomes a separate table

Relationship Sets
- connect two (or sometimes more) entities
- we can either:
  - create a table for the relationship ... with entity keys as foreign keys
  - add a foreign key to an entity table

For many-to-many relationships

- create a new table to represent the relationship
- for example:
  TeamMember(ssn, num)
- with two foreign keys:
  Employee(ssn, name, lot)  
  TeamMember.ssn → Employee.ssn
  Team(num, name, start)  
  TeamMember.num → Team.num
For **one-to-many relationships**

- include a foreign key to the “many” side of the relationship
- for example:

  Department(`code`, `name`)
  Employee(`ssn`, `name`, `lot`, `homedept`)

  where Employee.homedept → Department.code

**Alternatively, for one-to-many relationships**

- create a new table (as in many-to-many)
- for example:

  HomeDepartment(`ssn`, `code`)
  ... note that ssn is the key!
  Department(`code`, `name`)
  Employee(`ssn`, `name`, `lot`)

**What are the trade-offs between these two approaches?**

- Joins
- In new table, store more information (extra ssn in HomeDepartment)
What if the relationship set has attributes?

- if many-to-many, store in the new relationship table
- if one-to-many, store in the table where relationship is represented
Participation constraints in SQL

Enforce a required entity in a relationship using a foreign key

- using NOT NULL constraint

```
CREATE TABLE Department
(
    code INTEGER,
    name VARCHAR(20),
    manager_ssn VARCHAR(9) NOT NULL,
    start DATE,
    end DATE,
    PRIMARY KEY (code),
    FOREIGN KEY (manager_ssn) REFERENCES Employee (ssn)
) Engine=InnoDB;
```
Q: How should weak entity sets be translated?

- into a single table (recall 1..1 constraint)
- key of the strong entity is the foreign key
- key of the new table is key of the strong entity plus the partial key
- when strong entity is deleted, all weak entities also deleted

CREATE TABLE DependencyPolicy
(
    name VARCHAR(20),
    dob VARCHAR(10),
    ssn VARCHAR(11) NOT NULL,
    PRIMARY KEY (name, ssn),
    FOREIGN KEY (ssn) REFERENCES Employee (ssn) ON DELETE CASCADE
) Engine=InnoDB;
Summary of Translation [Elmasri & Navathe]

1. Create table and choose key for each entity set (include its attributes)

2. Create table for each weak entity set, include attributes and the key of the owner as a foreign key. Set the key as foreign key plus partial key.

3. For each one-to-one relationship set, add a foreign key to one of the entity sets involved in the relationship (a foreign key to the other entity in the relationship). Foreign key should be made unique (also a key).

4. For each one-to-many relationship set, add a foreign key to the entity set on the many side of the relationship (to reference the entity set on the one side of the relationship).

5. For each many-to-many relationship set, create a new table. Include a foreign key for each participant entity set in the relationship set. The key for the new table is the set of all such foreign keys.

Note on Normalization ...

- Only needed if for an entity set there are non-trivial, non-key FDs
- Can then normalize these after translation