CPSC 421
Database Management Systems

Lecture 7:
More on Database Design

* Some material adapted from R. Ramakrishnan, L. Delcambre, and B. Ludaescher

Today’s Agenda

• More on Database Design
  – Ternary relationships
  – Entities versus Attributes
  – Converting ER diagrams to relational schemas

• Quiz

• Start on normalization (introduction)
Ternary versus Binary Relationships

These two schemas are not equivalent!
When would we use a ternary relationship set?
When would we use three binary relationship sets?

This ternary relationship set means that a supplier must be authorized to supply a particular part to a particular project
For example
– “office depot” can supply “printer paper” to “project 112”
– “office max” can supply “paper clips” to “project 115”
– but this does not imply “office max” can supply “paper clips” to “112”
Ternary versus Binary Relationships

This ternary relationship set means that a supplier must be authorized to supply a particular part to a particular project.

For example:
- "office depot" can supply "printer paper" to "project 112"
- "office max" can supply "paper clips" to "project 115"
- but this does not imply "office max" can supply "paper clips" to "112"

The three binary relationship sets each represent something distinct:
- a supplier can be provide certain products (office max can provide pencils)
- a project can require certain products (112 requires pencils)
- a supplier can be authorized to use a certain supplier (112 is authorized to use office max)
- therefore, we might assume that office-max supplies pencils to 112
Ternary versus Binary Relationships

The three binary relationship sets each represent something distinct:

- A supplier can be provide certain products (office max can provide pencils)
- A project can require certain products (112 requires pencils)
- A supplier can be authorized to use a certain supplier (112 is authorized to use office max)
- Therefore, we might assume that office-max supplies pencils to 112

Duality: Entity vs Value / Attribute vs Relationship

Should office be an attribute of Employee?
Should office be a separate Entity Set?

- Most attributes can be “promoted” to an Entity Set … and some Entities can be “demoted” to an attribute value
- This is one reason why there are so many different ways to design a schema
What are some reasons to model Office as an Entity Set?

- An employee can have more than one office
- There are other attributes of Office
- Office needs to participate in other relationship sets
  - E.g., a relationship set connecting to furniture or telephones or network drops (located in the office)

Example when Office should be an entity set

- Office needs to participate in other relationship sets
Key Constraints as Described in the Textbook

Limiting participation in a relationship set to at most 1 entity
- Same as maximum multiplicity of 1 (x..1 in our notation)
- Each Department has at most one manager, according to the “key constraint” on Manages

Participation Constraints as in the Textbook

Every entity must participate in a relationship set
- Same as maximum multiplicity of 1 (1..x in our notation)
- Each Project must have at least one manager, according to the “participation constraint” on Project
Translating ERDs to Relational Schemas

Employee(ssn, name, lot, home-dept)
ProjectMember(ssn, number)
Department(code, name, manager)
Project(number, name, startdate, enddate, budget, sponsor)

This mapping assumes:
• Employees have one home department
• Departments have one manager
• Employees can participate in many projects

Translating Relationship Sets

• For relationship sets
  – we must indicate which entities we want to have connected
  – e.g., we need the key values for employees and teams stored together to represent the relationship
  – these could be in an existing table that represents one of the involved entities …
  – … or in a new table introduced explicitly to represent the relationship
Translating Relationship Sets

• For many-to-many relationship sets
  – create a new table to represent the relationship
  – For example:
    TeamMember\( (ssn, num) \)
    … with two foreign keys
    Employee\( (ssn, \text{name}, \text{lot}) \)
    Team\( (num, \text{name}, \text{start}) \)

Translating Relationship Sets

• For one-to-many relationship sets
  – introduce a foreign key to the “many” side of the relationship
  – For example:
    Department\( (\text{code}, \text{name}) \)
    Employee\( (ssn, \text{name}, \text{lot}, \text{homedept}) \)
    … where \text{homedept} is a foreign key (referencing Department \text{code})
Translating Relationship Sets

• Alternatively, for one-to-many relationship sets
  – Create a new table (like in many-to-many relationships)
  – For example:
    HomeDepartment(ssn, code) … note that ssn is the key
    Department(code, name)
    Employee(ssn, name, lot)

What are the tradeoffs between these approaches? …

Translating Relationship Sets

• Creating a new table from one-to-many relationships
  – requires a Join to obtain an employees home department
  – Each Employee ssn value associated with a home department is stored twice
  – … in the Employee and HomeDepartment table
Translating Relationship Sets

• When a many-many relationship set has attributes
  – Put them in the table that represents the relationship

  ProjectTeam(num, ssn, role, startdate, enddate)
  Project(num, name, startdate, enddate, budget)
  Employee(ssn, name, office)

Translating Relationship Sets

• When a one-to-many relationship set has attributes
  – Add them to the table where the relationship is represented

  Department(code, name, manager, startdate, enddate)
  Employee(ssn, name, office, dept)
  or else ...
  Manager(code, ssn, startdate, enddate)
Participation Constraints in SQL

- We can **require** any table to be in a binary relationship using a foreign key
  - by constraining the attribute to be **NOT NULL**

```sql
CREATE TABLE Department (
    code INTEGER,
    name VARCHAR(20),
    manager VARCHAR(9) NOT NULL,
    startdate DATE,
    enddate DATE,
    PRIMARY KEY (code),
    FOREIGN KEY (manager) REFERENCES Employee,
    ON DELETE NO ACTION );
```

Translating Weak Entity Sets

- Weak entity sets and identifying relationship sets are translated into a **single table**
- We must include the **key of the strong entity** as a foreign key
- The key for the table is the **key of the strong entity plus the partial key**
- When the **owner entity** is deleted, all owned weak entities must also be deleted

```sql
CREATE TABLE Policy (
    name VARCHAR(20),
    age INTEGER,
    ssn VARCHAR(11) NOT NULL,
    PRIMARY KEY (name, ssn) REFERENCES Employee,
    ON DELETE CASCADE )
```
Summary of the Translation [Elmasri & Navathe]

1. Create table and choose key for each entity set, include (single-valued) attributes
2. Create table for each weak entity set, include (single-valued) 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)*
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.
6. For each multi-valued attribute, construct a separate table. Repeat the key for the entity in this new table. It serves as both a key for this table and a foreign key to the original table for the entity.

* Unless the relationship set has attributes, in which case create a new table for the relationship set

Normalization

“Normalization” is the process of replacing a table with two or more tables

- For example, consider this schema:

  EmpDept

<table>
<thead>
<tr>
<th>EID</th>
<th>Name</th>
<th>Dept</th>
<th>DeptName</th>
</tr>
</thead>
<tbody>
<tr>
<td>A01</td>
<td>Ali</td>
<td>12</td>
<td>CS</td>
</tr>
<tr>
<td>A12</td>
<td>Eric</td>
<td>10</td>
<td>HR</td>
</tr>
<tr>
<td>A13</td>
<td>Eric</td>
<td>12</td>
<td>CS</td>
</tr>
<tr>
<td>A03</td>
<td>Anne</td>
<td>12</td>
<td>CS</td>
</tr>
</tbody>
</table>

- Versus these:

  Emp

<table>
<thead>
<tr>
<th>EID</th>
<th>Name</th>
<th>DeptID</th>
</tr>
</thead>
<tbody>
<tr>
<td>A01</td>
<td>Ali</td>
<td>12</td>
</tr>
<tr>
<td>A12</td>
<td>Eric</td>
<td>10</td>
</tr>
<tr>
<td>A13</td>
<td>Eric</td>
<td>12</td>
</tr>
<tr>
<td>A03</td>
<td>Anne</td>
<td>12</td>
</tr>
</tbody>
</table>

  Dept

<table>
<thead>
<tr>
<th>DeptID</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>HR</td>
</tr>
<tr>
<td>12</td>
<td>CS</td>
</tr>
</tbody>
</table>

- Which schema do you think is better? Why?
Normalization Issues

- The EmpDept schema combines two different concepts
  - Employee information, together with
  - Department information
- What is wrong with this?
  - If we separate the two concepts we could save space but some queries would run slower (Joins)
  - If we combine the two ideas we have redundancy but some queries would run faster (no Joins)

So we have a tradeoff …

Redundancy has a side effect: “anomalies”

Types of Anomalies

<table>
<thead>
<tr>
<th>EID</th>
<th>Name</th>
<th>Dept</th>
<th>DeptName</th>
</tr>
</thead>
<tbody>
<tr>
<td>A01</td>
<td>Ali</td>
<td>12</td>
<td>CS</td>
</tr>
<tr>
<td>A12</td>
<td>Eric</td>
<td>10</td>
<td>HR</td>
</tr>
<tr>
<td>A13</td>
<td>Eric</td>
<td>12</td>
<td>CS</td>
</tr>
<tr>
<td>A03</td>
<td>Anne</td>
<td>12</td>
<td>CS</td>
</tr>
</tbody>
</table>

“Update Anomaly”: If the CS department changes its name, we must change multiple rows in EmpDept

“Insertion Anomaly”: If a department has no employees, where do we store its id and name?

“Deletion Anomaly”: If A12 quits, the information about the HR department will be lost

- These are in addition to redundancy in general
  - For example, the department name is stored multiple times
Using NULL values can help insertion and deletion anomalies

- But NULL values have their own issues
  - They make aggregate operators harder to use
  - Not always clear what NULL means (does not apply, unknown, know but missing, see Melton podcast!)
  - May need outer joins instead of ordinary joins
  - In this case, EID is a primary care, and so it cannot contain a NULL value!
- They don’t address update anomalies or redundancy issues

### Decomposition

Normalization involves decomposing (partitioning) the table into separate tables

- Check to see if redundancy still exists (... repeat)

The key to understanding when and how to decompose schemas is through “functional dependencies”
  - which generalizes the notion of keys
For Thursday

• Reading
  – Ch 3: 3.5

• Be sure to understand:
  – Binary versus ternary relationships
  – How to convert from ER models to relational schemas
  – Next time we will talk about normalization

• Homework
  – Project proposal due on Thursday
  – Homework 3 due next Thursday