CMSC424: Database Design
E/R; Normalization

February 24, 2020

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

- Entity-Relationship Model Review
- Converting from E/R Model to Relational Schema
- Normalization
In Figure 7.15, we show an E-R diagram that corresponds to the university enterprise that we have been using thus far in the text. This E-R diagram is equivalent to the textual description of the university E-R model that we saw in Section 7.4, but with several additional constraints, and section now being a weak entity.

In our university database, we have a constraint that each instructor must have exactly one associated department. As a result, there is a double line in Figure 7.15 between instructor and inst_dept, indicating total participation of instructor in inst_dept; this is, each instructor must have associated department. Furthermore, there is an arrow from inst_dept to department, indicating that each instructor can have at most one associated department.
Thoughts...

- Nothing about actual data
  - How is it stored?

- No talk about the query languages
  - How do we access the data?

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

(a)

(b)

Figure 7.17 Alternatives for adding phone to the instructor entity set.
Entity sets vs Relationship sets
- Consider *takes*

![Diagram](image_url)

**Figure 7.18** Replacement of *takes* by *registration* and two relationship sets
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
Today’s Plan

- Entity-Relationship Model Review
- Converting from E/R Model to Relational Schema
-Normalization
Convert entity sets into a relational schema with the same set of attributes

- **Student**
  - ID
  - name
  - tot_cred

- **Instructor**
  - ID
  - name
  - salary

- **Student** (ID, name, tot_cred)
- **Instructor** (ID, name, salary)
Convert relationship sets *also* into a relational schema.

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

E/R Diagrams → Relations

- Convert relationship sets *also* into a relational schema.
- Remember: A relationship is completely described by primary keys of associate entities and its own attributes.

Advisor (student_ID, instructor_ID, date)

We can do better for many-to-one or one-to-one.
E/R Diagrams → Relations

Fold into Student:
Student(ID, name, tot_credits, advisor_ID)

Fold into Instructor:
Instructor(ID, name, salary, advisee_ID)
Figure 7.7: E-R diagram corresponding to instructors and students.

- Rectangles divided into two parts represent entity sets. The first part, which in this textbook is shaded blue, contains the name of the entity set. The second part contains the names of all the attributes of the entity set.
- Diamonds represent relationship sets.
- Undivided rectangles represent the attributes of a relationship set. Attributes that are part of the primary key are underlined.
- Lines link entity sets to relationship sets.
- Dashed lines link attributes of a relationship set to the relationship set.
- Double lines indicate total participation of an entity in a relationship set.
- Double diamonds represent identifying relationship sets linked to weak entity sets (we discuss identifying relationship sets and weak entity sets later, in Section 7.5.6).

Consider the E-R diagram in Figure 7.7, which consists of two entity sets, instructor and student related through a binary relationship set advisor. The attributes associated with instructor are ID, name, and salary. The attributes associated with student are ID, name, and tot_cred.

In Figure 7.7, attributes of an entity that are members of the primary key are underlined.

If a relationship set has some attributes associated with it, then we enclose the attributes in a rectangle and link the rectangle with a dashed line to the diamond representing that relationship set. For example, in Figure 7.8, we have the date descriptive attribute attached to the relationship set advisor to specify the date on which an instructor became the advisor.

Fold into Student:
Student(ID, name, tot_credits, advisor_ID)

OR

Fold into Instructor:
Instructor(ID, name, salary, advisee_ID)
Weak Entity Sets

Need to copy the primary key from the strong entity set:

Section(course_id, sec_id, semester, year)

Primary key for section = Primary key for course + Discriminator Attributes
## Multi-valued Attributes

The relationship set consists of the three entity sets `{phone_number, age()}`. Figure 7.11 also illustrates a multivalued attribute that corresponds to the role indicators Figure 7.12 depicts the role indicators whose component attributes are visible.

### instructor (ID, first_name, middle_name, last_name, street_number, street_name, apt_number, city, state, zip_code, date_of_birth)

### BUT

Phone_number needs to be split out into a separate table.

Instructor_Phone(Instructor_ID, phone_number)
Specialization and Generalization

A few different ways to handle it

1. Common table for common information and separate tables for additional information

   person (ID, name, street, city)
   employee (ID, salary)
   student (ID, tot_credits)

2. Separate tables altogether – good idea if an employee can’t be a student also – querying becomes harder (have to do unions for queries across all “persons”)

   employee (ID, name, street, city, salary)
   student (ID, name, street, city, tot_creds)
Today’s Plan

- Entity-Relationship Model Review
- Converting from E/R Model to Relational Schema
- Normalization
Relational Database Design

- Where did we come up with the schema that we used?
  - E.g. why not store the actor names with movies?

- If from an E-R diagram, then:
  - Did we make the right decisions with the E-R diagram?

- Goals:
  - Formal definition of what it means to be a “good” schema.
  - How to achieve it.
Movies Database Schema

Movie(\textit{title, year}, length, inColor, studioName, producerC#)
StarsIn(movieTitle, movieYear, starName)
MovieStar(\textit{name}, address, gender, birthdate)
MovieExec(name, address, cert#, netWorth)
Studio(\textit{name}, address, presC#)

Changed to:

Movie(\textit{title, year}, length, inColor, studioName, producerC#, \textit{starName})
<StarsIn merged into above>
MovieStar(\textit{name}, address, gender, birthdate)
MovieExec(name, address, \textit{cert#}, netWorth)
Studio(\textit{name}, address, presC#)

Is this a good schema ??
Movie(*title*, *year*, length, inColor, studioName, producerC#, *starName*)

<table>
<thead>
<tr>
<th>Title</th>
<th>Year</th>
<th>Length</th>
<th>inColor</th>
<th>StudioName</th>
<th>prodC#</th>
<th>StarName</th>
</tr>
</thead>
<tbody>
<tr>
<td>Star wars</td>
<td>1977</td>
<td>121</td>
<td>Yes</td>
<td>Fox</td>
<td>128</td>
<td>Hamill</td>
</tr>
<tr>
<td>Star wars</td>
<td>1977</td>
<td>121</td>
<td>Yes</td>
<td>Fox</td>
<td>128</td>
<td>Fisher</td>
</tr>
<tr>
<td>Star wars</td>
<td>1977</td>
<td>121</td>
<td>Yes</td>
<td>Fox</td>
<td>128</td>
<td>H. Ford</td>
</tr>
<tr>
<td>King Kong</td>
<td>2005</td>
<td>187</td>
<td>Yes</td>
<td>Universal</td>
<td>150</td>
<td>Watts</td>
</tr>
<tr>
<td>King Kong</td>
<td>1933</td>
<td>100</td>
<td>no</td>
<td>RKO</td>
<td>20</td>
<td>Fay</td>
</tr>
</tbody>
</table>

**Issues:**

1. Redundancy ➔ higher storage, inconsistencies (“anomalies”)
   
   *update anomalies, insertion anomalies*

2. Need nulls
   
   Unable to represent some information without using nulls

*How to store movies w/o actors (pre-productions etc) ?*
Movie(\textit{title}, \textit{year}, length, inColor, studioName, producerC\#, \textit{starNames})

<table>
<thead>
<tr>
<th>Title</th>
<th>Year</th>
<th>Length</th>
<th>inColor</th>
<th>StudioName</th>
<th>prodC#</th>
<th>StarNames</th>
</tr>
</thead>
<tbody>
<tr>
<td>Star wars</td>
<td>1977</td>
<td>121</td>
<td>Yes</td>
<td>Fox</td>
<td>128</td>
<td>{Hamill, Fisher, H. ford}</td>
</tr>
<tr>
<td>King Kong</td>
<td>2005</td>
<td>187</td>
<td>Yes</td>
<td>Universal</td>
<td>150</td>
<td>Watts</td>
</tr>
<tr>
<td>King Kong</td>
<td>1933</td>
<td>100</td>
<td>no</td>
<td>RKO</td>
<td>20</td>
<td>Fay</td>
</tr>
</tbody>
</table>

\textbf{Issues:}

3. Avoid sets
   - Hard to represent
   - Hard to query
**Smaller schemas always good ????**

Split Studio\((name, address, \text{presC}#)\) into:

**Studio1** \((name, \text{presC}#)\)  
**Studio2** \((name, address)\)

<table>
<thead>
<tr>
<th>Name</th>
<th>\text{presC}#</th>
<th>Address</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fox</td>
<td>101</td>
<td>Address1</td>
</tr>
<tr>
<td>Studio2</td>
<td>101</td>
<td>Address1</td>
</tr>
<tr>
<td>Universial</td>
<td>102</td>
<td>Address2</td>
</tr>
</tbody>
</table>

This process is also called “decomposition”

**Issues:**

4. Requires more joins (w/o any obvious benefits)

5. Hard to check for some dependencies

What if the “address” is actually the \text{presC}#’s address?

No easy way to ensure that constraint (w/o a join).
**Issues:**

6. “joining” them back results in more tuples than what we started with 
   (King Kong, 1933, Watts) & (King Kong, 2005, Faye)

   This is a “lossy” decomposition

   We lost some constraints/information

   The previous example was a “lossless” decomposition.
Desiderata

- No sets
- Correct and faithful to the original design
  - Avoid lossy decompositions
- As little redundancy as possible
  - To avoid potential anomalies
- No “inability to represent information”
  - Nulls shouldn’t be required to store information
- Dependency preservation
  - Should be possible to check for constraints

Not always possible. We sometimes relax these for: *simpler schemas, and fewer joins during queries.*
Some of Your Questions

- **Atomicity**
  - It depends primarily on how you use it
  - A String is not really atomic (can be split into letters), but do you want to query the letters directly? Or would your queries operate on the strings?

- **Which NF to use?**
  - Your choice – Normalization theory is a tool to help you understand the tradeoffs

- **Normal forms higher than 3NF?**
  - Actually we always use 4NF – we will discuss later

- **Trivial FDs**
  - Just means that: RHS is contained in LHS – that’s all
Approach

1. We will encode and list all our knowledge about the schema
   ◦ Functional dependencies (FDs)
     
     \[ \text{SSN} \rightarrow \text{name} \]  
     (means: \text{SSN} “implies” length)
   ◦ If two tuples have the same “SSN”, they must have the same “name”
     \[ \text{movietitle} \rightarrow \text{length} \]  
     ???? Not true.
   ◦ But, \[(\text{movietitle}, \text{movieYear}) \rightarrow \text{length}\] --- True.

2. We will define a set of rules that the schema must follow to be considered good
   ◦ “Normal forms”: 1NF, 2NF, 3NF, BCNF, 4NF, ...
   ◦ A normal form specifies constraints on the schemas and FDs

3. If not in a “normal form”, we modify the schema
## FDs: Example 1

<table>
<thead>
<tr>
<th>Title</th>
<th>Year</th>
<th>Length</th>
<th>StarName</th>
<th>Birthdate</th>
<th>producerC#</th>
<th>Producer - address</th>
<th>Prdocuer - name</th>
<th>netWorth</th>
</tr>
</thead>
<tbody>
<tr>
<td>Plane Crazy</td>
<td>1927</td>
<td>6</td>
<td>NULL</td>
<td>NULL</td>
<td>WD100</td>
<td>Mickey Rd</td>
<td>Walt Disney</td>
<td>100000</td>
</tr>
<tr>
<td>Star Wars</td>
<td>1977</td>
<td>121</td>
<td>H. Ford</td>
<td>7/13/42</td>
<td>GL102</td>
<td>Tatooine</td>
<td>George Lucas</td>
<td>10^9</td>
</tr>
<tr>
<td>Star Wars</td>
<td>1977</td>
<td>121</td>
<td>M. Hamill</td>
<td>9/25/51</td>
<td>GL102</td>
<td>Tatooine</td>
<td>George Lucas</td>
<td>10^9</td>
</tr>
<tr>
<td>Star Wars</td>
<td>1977</td>
<td>121</td>
<td>C. Fisher</td>
<td>10/21/56</td>
<td>GL102</td>
<td>Tatooine</td>
<td>George Lucas</td>
<td>10^9</td>
</tr>
<tr>
<td>King Kong</td>
<td>1933</td>
<td>100</td>
<td>F. Wray</td>
<td>9/15/07</td>
<td>MC100</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>King Kong</td>
<td>2005</td>
<td>187</td>
<td>N. Watts</td>
<td>9/28/68</td>
<td>PJ100</td>
<td>Middle Earth</td>
<td>Peter Jackson</td>
<td>10^8</td>
</tr>
</tbody>
</table>
## FDs: Example 2

<table>
<thead>
<tr>
<th>State Name</th>
<th>State Code</th>
<th>State Population</th>
<th>County Name</th>
<th>County Population</th>
<th>Senator Name</th>
<th>Senator Elected</th>
<th>Senator Born</th>
<th>Senator Affiliation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Alabama</td>
<td>AL</td>
<td>4779736</td>
<td>Autauga</td>
<td>54571</td>
<td>Jeff Sessions</td>
<td>1997</td>
<td>1946</td>
<td>‘R’</td>
</tr>
<tr>
<td>Alabama</td>
<td>AL</td>
<td>4779736</td>
<td>Baldwin</td>
<td>182265</td>
<td>Jeff Sessions</td>
<td>1997</td>
<td>1946</td>
<td>‘R’</td>
</tr>
<tr>
<td>Alabama</td>
<td>AL</td>
<td>4779736</td>
<td>Barbour</td>
<td>27457</td>
<td>Jeff Sessions</td>
<td>1997</td>
<td>1946</td>
<td>‘R’</td>
</tr>
<tr>
<td>Alabama</td>
<td>AL</td>
<td>4779736</td>
<td>Autauga</td>
<td>54571</td>
<td>Richard Shelby</td>
<td>1987</td>
<td>1934</td>
<td>‘R’</td>
</tr>
<tr>
<td>Alabama</td>
<td>AL</td>
<td>4779736</td>
<td>Baldwin</td>
<td>182265</td>
<td>Richard Shelby</td>
<td>1987</td>
<td>1934</td>
<td>‘R’</td>
</tr>
<tr>
<td>Alabama</td>
<td>AL</td>
<td>4779736</td>
<td>Barbour</td>
<td>27457</td>
<td>Richard Shelby</td>
<td>1987</td>
<td>1934</td>
<td>‘R’</td>
</tr>
</tbody>
</table>
## FDs: Example 3

<table>
<thead>
<tr>
<th>Course ID</th>
<th>Course Name</th>
<th>Dept Name</th>
<th>Credits</th>
<th>Section ID</th>
<th>Semester</th>
<th>Year</th>
<th>Building</th>
<th>Room No.</th>
<th>Capacity</th>
<th>Time Slot ID</th>
</tr>
</thead>
</table>

Functional dependencies

- course_id $\rightarrow$ title, dept_name, credits, building, room_number $\rightarrow$ capacity
- course_id, section_id, semester, year $\rightarrow$ building, room_number, time_slot_id
Examples from Quiz

- advisor(s_id, i_id, s_name, s_dept_name, i_name, i_dept_name)

- friends(userid1, userid2, name1, name2, birthdate1, birthdate2)