Outline

- Relational Algebra (6.1)
- E/R Model (7.2 - 7.4)
- E/R Diagrams (7.5)
- Reduction to Schema (7.6)
- Relational Database Design (7.7)
- Functional Dependencies (8.1 – 8.4)
- Normalization (8.5 – 8.7)
- Relational Query Languages
- SQL Basics
- Formal Semantics of SQL

E-R Diagram for a University Enterprise
Redundant Attributes

- Suppose we have entity sets
  - instructor, with attributes including dept_name
  - department
  and a relationship
  - inst_dept relating instructor and department
- Attribute dept_name in entity instructor is redundant since there is an explicit relationship inst_dept which relates instructors to departments
  - The attribute replicates information present in the relationship, and should be removed from instructor
  - BUT: redundant attributes sometimes get reintroduced when converting back to tables

Reduction to Relation Schemas

- Entity sets and relationship sets can be expressed uniformly as relation schemas that represent the contents of the database.
  - A database which conforms to an E-R diagram can be represented by a collection of schemas.
- For each entity set and relationship set there is a unique schema that is assigned the name of the corresponding entity set or relationship set.
- Each schema has a number of columns (generally corresponding to attributes), which have unique names.
Representing Entity Sets

- A strong entity set reduces to a schema with the same attributes
  \( \text{course(} \text{course_id, title, credits)} \) 

- A weak entity set becomes a table that includes a foreign key for the primary key of the identifying strong entity set
  \( \text{section(} \text{course_id, sec_id, sem, year)} \) 

Representing Relationship Sets

- A many-to-many relationship set is represented as a schema with attributes for the primary keys of the two participating entity sets, and any descriptive attributes of the relationship set.
- Example: schema for relationship set \( \text{advisor} \):
  \( \text{advisor(s_id, i_id)} \)
Redundancy of Schemas

• Many-to-one and one-to-many relationship sets that are total on the many-side can be represented by adding an extra attribute(s) to the “many” side, containing the primary key of the “one” side

• Example:
  
  • get rid of inst_dept relationship set by:
    
    instructor(ID, name, salary) → instructor(ID, dept_name, name, salary)

Redundancy of Schemas (Cont.)

• For one-to-one relationship sets, either side can be chosen to act as the “many” side

  • That is, extra attribute can be added to either of the tables corresponding to the two entity sets

  instructor(ID, name, salary)

  or

  department(dept_name, ID, building, budget)
Redundancy of Schemas (Cont.)

• If participation is partial on the “many” side, replacing a relationship schema by an extra attribute in the schema corresponding to the “many” side could result in null values (generally avoided)
  • i.e. the approach in the previous slides does not work
  • need to represent relationship as a separate table
  • Unless otherwise instructed, assume we wish to avoid NULLs when converting to relations, i.e. remove redundant relationship schemas only when total participation on side where adding attribute.

• Relationship set linking a weak entity set to its identifying strong entity set is redundant.
  • Example: The section schema contains all attributes that would be in sec_course

Composite Attributes

<table>
<thead>
<tr>
<th>instructor</th>
</tr>
</thead>
<tbody>
<tr>
<td>ID</td>
</tr>
<tr>
<td>name</td>
</tr>
<tr>
<td>first_name</td>
</tr>
<tr>
<td>middle_initial</td>
</tr>
<tr>
<td>last_name</td>
</tr>
<tr>
<td>address</td>
</tr>
<tr>
<td>street</td>
</tr>
<tr>
<td>street_number</td>
</tr>
<tr>
<td>street_name</td>
</tr>
<tr>
<td>apt_number</td>
</tr>
<tr>
<td>city</td>
</tr>
<tr>
<td>state</td>
</tr>
<tr>
<td>zip</td>
</tr>
<tr>
<td>phone_number</td>
</tr>
<tr>
<td>date_of_birth</td>
</tr>
<tr>
<td>age( )</td>
</tr>
</tbody>
</table>

• Composite attributes flattened out
  • Example: given entity set instructor
    • with composite attribute name
    • with component attributes first_name and last_name
    • replace with name_first_name and name_last_name
      - Prefix omitted if there is no ambiguity
  • Ignoring multivalued attributes, extended instructor schema is
    • instructor(ID,
      first_name, middle_initial, last_name,
      street_number, street_name,
      apt_number, city, state, zip_code,
      date_of_birth)
Multivalued Attributes

- Multivalued attribute $M$ of entity $E$ represented by a separate schema $EM$
  - Schema $EM$ includes $E$’s primary key and attribute corresponding to $M$
  - Example: Multivalued attribute `phone_number` of `instructor`:
    
    ```
    inst_phone = (ID, phone_number)
    ```

- Each value of the multivalued attribute maps to separate tuple of $EM$:
  - `instructor` entity with primary key 22222 and numbers 456-7890 and 123-4567 maps to:
    - (22222, 456-7890)
    - (22222, 123-4567)

Multivalued Attributes (Cont.)

- Special case: entity `time_slot` has only one attribute other than the primary-key attribute, and that attribute is multivalued
  - Optimization: Don’t create the relation corresponding to the entity, just create the one corresponding to the multivalued attribute
  - `time_slot(slot_id, day, start_time, end_time)`
ER Diagram to Relational Schema

- Schema per entity set
  - expand composite attributes
  - new schema for multi-valued
  - drop derived attributes for now
- Schema per relationship set

lots of foreign key dependences (weak, relationships..)

ER Diagram to Relational Schema

- Schema per entity set
  - expand composite attributes
  - new schema for multi-valued
  - drop derived attributes for now
- Schema per relationship set

lots of foreign key dependences (weak, relationships..)
ER Diagram to Relational Schema

- Schema per entity set
  - expand composite attributes
  - new schema for multi-valued
  - drop derived attributes for now
- Schema per relationship set

lots of foreign key dependences (weak, relationships..)

ER Diagram to Relational Schema

- Schema per entity set
  - expand composite attributes
  - new schema for multi-valued
  - drop derived attributes for now
- Schema per relationship set

lots of foreign key dependences (weak, relationships..)
ER Diagram to Relational Schema

- Schema per entity set
  - expand composite attributes
  - new schema for multi-valued
  - drop derived attributes for now

- Schema per relationship set

lots of foreign key dependences (weak, relationships..)

ER Diagram to Relational Schema

- Schema per entity set
  - expand composite attributes
  - new schema for multi-valued
  - drop derived attributes for now

- Schema per relationship set

lots of foreign key dependences (weak, relationships..)
ER Diagram to Relational Schema

- Schema per entity set
  - expand composite attributes
  - new schema for multi-valued
  - drop derived attributes for now
- Schema per relationship set

lots of foreign key dependences (weak, relationships..)

lots of foreign key dependences (weak, relationships..)
ER Diagram to Relational Schema

- Schema per entity set
  - expand composite attributes
  - new schema for multi-valued
  - drop derived attributes for now
- Schema per relationship set

lots of foreign key dependences (weak, relationships..)
Binary Vs. Non-Binary Relationships

• Some relationships that appear to be non-binary may be better represented using binary relationships
  
  – E.g., A ternary relationship parents, relating a child to his/her father and mother, is best replaced by two binary relationships, parent1 and parent2
    – Using two binary relationships allows partial information (e.g., only parent1 being known)
  
• But there are some relationships that are naturally non-binary
  – Example: proj_group, with several project members

Design Issues

• **Binary versus n-ary relationship sets**
  Although it is possible to replace any nonbinary (n-ary, for \( n > 2 \)) relationship set by a number of distinct binary relationship sets, a n-ary relationship set shows more clearly that several entities participate in a single relationship.

• **Placement of relationship attributes can be tricky**
  e.g., attribute date as attribute of advisor or as attribute of student
Converting Non-Binary Relationships to Binary Form

- In general, any non-binary relationship can be represented using binary relationships by creating an artificial entity set.
  - Replace $R$ between entity sets $A$, $B$ and $C$ by an entity set $E$, and $R_A$, $R_B$, $R_C$, relating $E$ with $A$, $B$, and $C$
  - Create a special identifying attribute for $E$
  - Add any attributes of $R$ to $E$
  - For each relationship $(a_i, b_i, c_i)$ in $R$
    - create a new entity $e_i$ in the entity set $E$
    - add $(e_i, a_i)$ to $R_A$, etc.

A Movie Industry Schema
Outline

- Relational Algebra (6.1)
- E/R Model (7.2 - 7.4)
- E/R Diagrams (7.5)
- Reduction to Schema (7.6)
- Relational Database Design (7.7)
- Functional Dependencies (8.1 – 8.4)
- Normalization (8.5 – 8.7)

A Movie Industry Schema
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.

Relational Schemas and Redundancy

- movies(name, year, len)
- stars(name, addr, gender, birthdate)
- execs(name, cert#)
- studios(stud_name, address)

- in(star_name, movie_name, movie_year)
- made_by(movie_name, movie_year, studioname)
- produced_by(movie_name, movie_year, cert#)
- helmed_by(cert#, stud_name)
Relational Schemas and Redundancy

- movies(name, year, len)
- stars(name, addr, gender, birthdate)
- execs(name, cert#)
- studios(stud_name, address, pres#)

- in(star_name, movie_name, movie_year)
- made_by(movie_name, movie_year, studioname)
- produced_by(movie_name, movie_year, cert#)
Relational Schemas and Redundancy

- movies(name, year, len, prod#, studio_name)
- stars(name, addr, gender, birthdate)
- execs(name, cert#)
- studios(stud_name, address, pres#)

- in(star_name, movie_name, movie_year)

Is this a good idea???
Relational Database Design

or

“Troubles With Schemas”

Exam 1

Covers:
• lectures
• quizzes 1-4, first five questions on quiz 5
• assignments 1-4

Answers visible on gradescope:
• quiz 2
• quiz 3
• quiz 4 (2/24)
• quiz 5 questions 1-5 (3/1)

Practice exams will be posted today
• Not inclusive
• Have some topics we will not cover

Short review march 1
Outline

- Relational Algebra (6.1)
- E/R Model (7.2 - 7.4)
- E/R Diagrams (7.5)
- Reduction to Schema (7.6)
- Relational Database Design (7.7)
- Functional Dependencies (8.1 – 8.4)
- Normalization (8.5 – 8.7)

---

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,
2. Inconsistencies (“anomalies”)
   - update anomalies, insertion anomalies
3. Need nulls!
   - movies w/o actors, pre-productions, etc
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>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>

**Issues:**

3. Avoid sets
   - Hard to represent
   - Hard to query

**Less Redundancy through Decomposition**

Split Studio($name$, address, $presC#$) into:
   - Studio1 ($name$, $presC#$),
   - Studio2($name$, address)???

<table>
<thead>
<tr>
<th>Name</th>
<th>$presC#$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fox</td>
<td>101</td>
</tr>
<tr>
<td>Studio2</td>
<td>101</td>
</tr>
<tr>
<td>Universal</td>
<td>102</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Name</th>
<th>Address</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fox</td>
<td>Address1</td>
</tr>
<tr>
<td>Studio2</td>
<td>Address1</td>
</tr>
<tr>
<td>Universal</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 $presC#$’s address?
   - No easy way to ensure that constraint (w/o a join).
Are smaller schemas always good ????

Decompose StarsIn(movieTitle, movieYear, starName) into:

StarsIn1(movieTitle, movieYear)
StarsIn2(movieTitle, starName) ???

<table>
<thead>
<tr>
<th>movieTitle</th>
<th>movieYear</th>
</tr>
</thead>
<tbody>
<tr>
<td>Star wars</td>
<td>1977</td>
</tr>
<tr>
<td>King Kong</td>
<td>1933</td>
</tr>
<tr>
<td>King Kong</td>
<td>2005</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>movieTitle</th>
<th>starName</th>
</tr>
</thead>
<tbody>
<tr>
<td>Star Wars</td>
<td>Hamill</td>
</tr>
<tr>
<td>King Kong</td>
<td>Watts</td>
</tr>
<tr>
<td>King Kong</td>
<td>Faye</td>
</tr>
</tbody>
</table>

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.