Today ...

- Quiz 3
- Normalization wrap up
- ERD intro

Homework

- HW 3 out
Properties of Decompositions

Basic idea of normalization

Decompose a table using the “bad” FDs $X \rightarrow Y$ by ...

- removing $Y$ from the original table
- creating a table out of $XY$
- making $X$ the new primary key for the $XY$ table

A “good” decomposition is considered:

- **Lossless** $\ldots$ we can get the original table back
- **Dependency Preserving** $\ldots$ can still enforce all of the FDs
- in BCNF (if possible) or else 3NF

Checking for Lossless Decompositions ...

- if relation $R(A)$ is decomposed into $R_1(A_1)$ and $R_2(A_2)$
- its lossless iff $A_1 \cap A_2$ contains a key in either $R_1$ or $R_2$

Lossless means we can get back the original relation before decomposition

- if two relations don’t share a key ...
- we could get back more rows than the original
Example 1:

- \text{EmpDept}(\text{eid}, \text{name}, \text{dept}) \text{ and } \text{Dept}(\text{dept}, \text{dept\_name})
- Q: Is this a “lossless decomposition”?

Example 2:

- \text{ZipCode}(\text{zip}, \text{state}) \text{ and } \text{Location}(\text{addr}, \text{city}, \text{zip})
- Q: Is this a “lossless decomposition”? ... YES!
- Q: Is this a “dependency preserving” decomposition?
  \begin{itemize}
  \item NO! ... \text{addr, city, state} \rightarrow \text{zip} \& \text{zip} \rightarrow \text{state}
  \end{itemize}
- For example:

<table>
<thead>
<tr>
<th>ZipCode</th>
<th>Location</th>
</tr>
</thead>
<tbody>
<tr>
<td>\text{zip}</td>
<td>\text{state}</td>
</tr>
<tr>
<td>99202 WA</td>
<td>42 1st Ave Spokane 99202</td>
</tr>
<tr>
<td>99203 WA</td>
<td>42 1st Ave Spokane 99203</td>
</tr>
</tbody>
</table>

- “Denormalizing” these gives ... i.e., undoing the decomposition

<table>
<thead>
<tr>
<th>addr</th>
<th>city</th>
<th>state</th>
<th>zip</th>
</tr>
</thead>
<tbody>
<tr>
<td>42 1st Ave Spokane WA 99202</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>42 1st Ave Spokane WA 99203</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- Which violates the original functional dependencies!!!

- So: BCNF but not a “good” decomposition (best we can do is 3NF)
More on Dependency Preservation ...

To determine if dependencies are preserved after decomposition
- we often have to “reason” over resulting dependencies

Example 1:

Original Table
- \( R(a, b, c, d, e) \)
- With FD’s: \( ab \rightarrow cde \) (key), \( b \rightarrow d \), and \( d \rightarrow e \)
- What NF is \( R \) in? \( \ldots 1NF \) because of \( b \rightarrow d \)

Decomposition
- \( R_1(a, b, c) \) \( \ldots \) what is the key?
- \( R_2(b, d) \) \( \ldots \) what is the key?
- \( R_3(d, e) \) \( \ldots \) what is the key?

Is this a dependency preserving decomposition?
- From \( R_1 \): \( ab \rightarrow c \)
- From \( R_2 \): \( b \rightarrow d \)
- From \( R_3 \): \( d \rightarrow e \)
- What about the original key? \( \ldots \)
  - \( ab \rightarrow c \) from \( ab \rightarrow c \) in \( R_1 \)
  - \( ab \rightarrow cd \) from \( b \rightarrow d \) in \( R_2 \)
  - \( ab \rightarrow cde \) from \( d \rightarrow e \) in \( R_3 \) \( \ldots \) via transitivity
Summary

1. Identify Functional Dependencies (FDs) in your relations
2. See if any of them are “bad” FDs (non-trivial, non-key)
3. Decompose to get rid of “bad” FDs
   - but watch out for dependencies to a part of a candidate key (3NF)
4. Check that your decomposition is lossless and dependency preserving

(*) Note that we have discussed these steps “informally” ...
Conceptual Data Modeling Approaches

Design typically involves multiple steps prior to creating tables!

- We'll focus on the **Entity-Relationship Model** (ER)
- Similar to UML diagrams in software design
- Sometimes called ER diagrams, or ERDs

![ER Diagram](image-url)

**Key:**
- **Entity Set**
- **Relationship Set**
- **Attribute**
**ER Terminology**

An “**Entity**”

- an object distinguishable from other objects
- e.g., the employee “John Smith”
- described using a set of attribute-value pairs
- designated attribute (or composite attributes) serves as an “identifier” (key)

An “**Entity Set”**

- a collection of similar entities
- defined by attributes and relationships that characterize the entities
- sometimes just called “Entity” (when it is clear we’re talking about the set)
- sometimes referred to as the **Entity Type**
- an entity is *an instance* (or member) of an entity set
A “Relationship”

- an association among 2 or more entities
- e.g., John Smith’s *home department* is Pharmacy 2

A “Relationship Set”

- a collection of similar relationships
- e.g., the set of *home department* relationships
- defined by the participating entity types and other constraints
- just called a “Relationship” (when context clear)
- also sometimes called a “Relationship Type”
- a relationship is *an instance* (or member) of a relationship set
**ER versus the Relational Model**

ER is a different data model than the relational model

- different *constructs* for modeling schemas and instances
- they are pretty close though

The **relational model** has ...

- tables (relations) with attributes, keys and foreign keys, rows, values

The **ER model** has ...

- entity & entity sets with attributes and entity identifiers (like keys)
- relationship & relationship sets with cardinality constraints, roles, attributes

Usually start with ER models, then map them to relational schemas