Normalization: Agenda

- Data Redundancy and Anomalies
- Functional Dependencies
  - Determinants
- First Normal Form (1NF), Second Normal Form (2NF), Third Normal Form (3NF)
- Boyce-Codd Normal Form (BCNF)
- Principles of Database design

Review

- Two ways of viewing a database:
  - As a set of relations, and the associated definitions of tuples, keys, etc.
    - Nothing to do with a real-life situation
  - An implementation of a real-life model, namely the E-R model
    - A one-to-one mapping between the elements in the model, and the elements of the database
The E-R Model

- Provided a structure to specify all the entities and their attributes that we are interested in
- Provided a common language between users and modelers that could be directly converted into a database
- ER diagrams – a diagrammatic representation of the ER model
- EER diagrams – because simple ER diagrams became too unwieldy in modern organizational contexts, where every entity is captured in databases

We remember once again that these diagrams, though extremely useful, are ultimately not what we are after.

The E-R Model

- When we convert EER diagrams to relations:
  - Some of the richness of the context is lost (although foreign keys do indicate relationships)
  - All we know is now in the form of some special tables or relations, that satisfy some criteria

Characteristics of relations

- Not all database tables qualify as relations (?)
Characteristics of relations

- Not all database tables qualify as relations
- Requirements (to be a relation):
  - Every relation has a unique name
  - Every attribute value is atomic
  - Every row is unique
  - Attributes (columns) in a given table have unique names
  - The order of the columns is irrelevant
  - The order of the rows is irrelevant

Is that all that needs to be done?

- NO!
- We still have to deal with the problems that made us consider relational databases in the first place
  - Redundancy
  - Various anomalies
  - Update anomalies
  - Insertion anomalies
  - Deletion anomalies
    - The “requirements” for relations (previous slide) are still too generic to take care of these problems
- All such problems make it harder to create, maintain or modify the database
- Solution: Normalization

Normalization

- **Defn**: A process for evaluating and correcting table structures to minimize data redundancies, thereby reducing the likelihood of data anomalies.
  - i.e., the process of decomposing relations with anomalies to produce smaller, well-structured relations
- First developed by E.F. Codd, since modified by others
- Normalization is a “check” against potential mistakes / inefficiencies that might be in the structure of the database tables
"Well-structured" relations

- A relation that contains minimal data redundancy and allows users to insert, delete, and update rows without causing data inconsistencies is "well-structured".
- Goal is to avoid anomalies:
  - Insertion Anomaly – adding new rows forces user to create duplicate data.
  - Deletion Anomaly – deleting rows may cause a loss of data that would be needed for other future rows.
  - Modification (Update) Anomaly – changing data in a row forces changes to other rows because of duplication.
- General rule of thumb: a table should not pertain to more than one entity type ("entity" being of course something we are interested in).

Anomalies revisited

<table>
<thead>
<tr>
<th>Snum</th>
<th>SName</th>
<th>SAddress</th>
<th>SPosition</th>
<th>Bnum</th>
<th>BTel_No</th>
<th>BFax_No</th>
</tr>
</thead>
<tbody>
<tr>
<td>S11</td>
<td>Jane Doe</td>
<td>11 Wood St.</td>
<td>Manager</td>
<td>B5</td>
<td>817-256-2234</td>
<td>817-256-2231</td>
</tr>
<tr>
<td>S23</td>
<td>Ann Martin</td>
<td>114 S. Main</td>
<td>Deputy</td>
<td>B4</td>
<td>972-456-8970</td>
<td>972-456-8842</td>
</tr>
<tr>
<td>S2</td>
<td>Leslie King</td>
<td>112 S. Main</td>
<td>Deputy</td>
<td>B4</td>
<td>972-456-8970</td>
<td>972-456-8842</td>
</tr>
<tr>
<td>S15</td>
<td>Matt Hoffa</td>
<td>29 Market St.</td>
<td>Assistant</td>
<td>B6</td>
<td>317-869-3141</td>
<td>317-869-1123</td>
</tr>
<tr>
<td>S45</td>
<td>Jill Emory</td>
<td>11 S. Elm</td>
<td>Manager</td>
<td>B4</td>
<td>972-456-8970</td>
<td>972-456-8842</td>
</tr>
</tbody>
</table>

STAFF/BRANCH relation

Exercise

- What happens if we want to insert details of a new staff member in branch B4?
- What happens if we want to insert information about a new branch?
- What happens if Jane Doe is fired?
- What happens if branch B4’s area code changes?
Functional dependency

- If \( A \) and \( B \) are two sets of attributes of a relation \( R \), then \( B \) is functionally dependent on \( A \) if each value of \( A \) in the relation is associated with exactly one value of \( B \).
  - Represented by \( A \rightarrow B \)
  - \( A \) is called the determinant
  - If we know the values of \( A \), the values of \( B \) are uniquely determined

- Examples:
  - Staff_Num \( \rightarrow \) Staff_Name
  - Emp_ID, Course_Title \( \rightarrow \) Date_Completed
  - SSN \( \rightarrow \) Name, Address, Birthdate

- Recall definition of candidate key: uniquely identifies the row of a relation
  - Implies that all non-key attributes are functionally dependent on the candidate key(s)

First Normal Form (1NF)

- Any relation that does not have multi-valued attributes is in First Normal Form
  - “Atomic”
- In fact, any valid relation is in 1NF, by definition

Example

<table>
<thead>
<tr>
<th>Multivalued attributes (PROJECT)</th>
<th>Multi-valued attributes (PROJECT)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Proj_Num Proj_Name Emp_Num Emp_Name Job_Class Salary</td>
<td></td>
</tr>
<tr>
<td>17 Greenpeace 12 Zoeller Elect. Engineer $50,000</td>
<td></td>
</tr>
<tr>
<td>17 Greenpeace 14 Woods Database Designer $45,000</td>
<td></td>
</tr>
<tr>
<td>17 Greenpeace 6 Lopez Systems Analyst $45,000</td>
<td></td>
</tr>
<tr>
<td>23 Allentown 12 Sorenstam Data Analyst $48,000</td>
<td></td>
</tr>
<tr>
<td>20 Monterey 5 Mickelson DSS Analyst $65,000</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Atomic (single-valued) attributes (PROJECT2)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Proj_Num Proj_Name Emp_Num Emp_Name Job_Class Salary</td>
</tr>
<tr>
<td>17 Greenpeace 12 Zoeller Elect. Engineer $50,000</td>
</tr>
<tr>
<td>17 Greenpeace 14 Woods Database Designer $45,000</td>
</tr>
<tr>
<td>23 Allentown 13 Sorenstam Database Designer $48,000</td>
</tr>
<tr>
<td>20 Monterey 5 Mickelson DSS Analyst $65,000</td>
</tr>
</tbody>
</table>
Are we there yet?

- **Insertion** – can’t enter a new employee without assigning him or her to a project
- **Deletion** – if we remove employee 5, we lose information about the existence of the Monterey project
- **Modification** – giving a salary increase to employee 6 forces us to update multiple records

- Why do these anomalies exist?
  - Because we’ve combined two entity types into one relation. This results in duplication, and an unnecessary dependency between the entities

Continuing normalization

- **Starting the normalization process:**
  1. Eliminate repeating groups (multi-valued attributes)
  2. Identify the Primary Key
  3. Identify all dependencies

So... What is the primary key? / What are the dependencies?

<table>
<thead>
<tr>
<th>Proj_Num</th>
<th>Proj_Name</th>
<th>Emp_Num</th>
<th>Emp_Name</th>
<th>Job_Class</th>
<th>Salary</th>
<th>Location</th>
</tr>
</thead>
<tbody>
<tr>
<td>17</td>
<td>Greenpeace</td>
<td>14</td>
<td>Zoeller</td>
<td>Elect. Engineer</td>
<td>$50,000</td>
<td>Vienna</td>
</tr>
<tr>
<td>12</td>
<td>Greenpeace</td>
<td>6</td>
<td>Lopez</td>
<td>Systems Analyst</td>
<td>$45,000</td>
<td>Reston</td>
</tr>
<tr>
<td>23</td>
<td>Monterey</td>
<td>6</td>
<td>Lopez</td>
<td>Systems Analyst</td>
<td>$48,000</td>
<td>Reston</td>
</tr>
<tr>
<td>25</td>
<td>Monterey</td>
<td>5</td>
<td>Meltonen</td>
<td>Proj. Analyst</td>
<td>$45,000</td>
<td>Vienna</td>
</tr>
</tbody>
</table>

Dependences in PROJECT2

- **Desirable dependencies:**
  - Proj_Num → Proj_Name
  - Emp_Num → Emp_Name, Job_Class, Salary

- **Less desirable dependencies:**
  - Transitive dependency:
    - Job_Class → Salary
Second Normal Form (2NF)

- A relation is in second normal form or 2NF if it is in first normal form, and every non-key attribute is \textit{fully} functionally dependent on the primary key.

- “Fully” implies no \textit{partial} functional dependencies, i.e. no non-key attributes are functionally dependent on only part of the primary key.

Converting to 2NF

- Starting with a relation in 1NF, do the following:
  1. Make new tables to eliminate partial dependencies
     - A new table is needed for each component of the primary key that is a determinant in a partial dependency
     - The original (composite) primary key should remain in the original table
  2. Reassign the corresponding dependent attributes
     - Remove dependent attributes from the original table and move them to the corresponding new table

- Note: any relation in 1NF that has a single-attribute primary key, is already automatically in 2NF.

Converting the PROJECT2 data to 2NF

<table>
<thead>
<tr>
<th>New table Name</th>
<th>PROJECT</th>
<th>Partial dependencies:</th>
</tr>
</thead>
<tbody>
<tr>
<td>Proj_Num</td>
<td>Proj_Name</td>
<td></td>
</tr>
<tr>
<td>Emp_Num</td>
<td>Emp_Name, Job_Class, Salary</td>
<td></td>
</tr>
</tbody>
</table>

All converted to full functional dependencies.
Third Normal Form (3NF)

• A relation is in third normal form, or 3NF, if it is in second normal form and it contains no transitive dependencies

• Converting to 3NF:
  1. Make new tables to eliminate transitive dependencies
     – The determinant of each transitive dependency becomes the primary key of its own new table
  2. Reassign corresponding dependent attributes
     – Remove dependent attributes from the original table and move them to the corresponding new table

Converting the PROJECT2 data to 3NF

Table Name: PROJECT
- Prof_Name
- Prof_Name

Table Name: EMPLOYEE
- Emp_Num
- Emp_Name
- Job_Class

Table Name: ASSIGNMENT
- Prof_Name
- Emp_Num
- Location

New table name: JOB
- Job_Class
- Salary

Transitive dependency: Job_Class → Salary

Boyce-Codd Normal Form (BCNF)

• What if a non-key attribute is the determinant of a key attribute?
  • Doesn’t violate 3NF
    • Transitive dependencies are defined to be from one non-key attribute to another non-key attribute
    • This is different:
Non-BCNF example

- The business rule (relationship) causing the problem is that each class is taught by exactly one professor….

Examples of potential problems:
- (update anomaly) What if a new professor is assigned to 91952?
- (insertion anomaly) What if a professor is assigned a new class?
- (deletion anomaly) If student 143 withdraws, what happens to professor 17?

<table>
<thead>
<tr>
<th>Stu_ID</th>
<th>Prof_ID</th>
<th>Class_Code</th>
<th>Enroll_Grade</th>
</tr>
</thead>
<tbody>
<tr>
<td>125</td>
<td>12</td>
<td>90971</td>
<td>A</td>
</tr>
<tr>
<td>143</td>
<td>17</td>
<td>90023</td>
<td>B</td>
</tr>
<tr>
<td>191</td>
<td>12</td>
<td>90279</td>
<td>A</td>
</tr>
<tr>
<td>191</td>
<td>15</td>
<td>91952</td>
<td>A</td>
</tr>
</tbody>
</table>

Boyce-Codd Normal Form (cont.)

- A table is in Boyce-Codd normal form, or BCNF, if every determinant in the table is a candidate key

When are 3NF and BCNF the same?

Principles for good database design

- Relations in a well designed database meet the following criteria
  - No redundancy
  - No partial dependencies
  - No transitive dependencies
- Guidelines
  - Identify entities involved and their relevant attributes and identifiers
  - Define relationships between entities
  - Draw an ER / EER diagram to model the problem
  - Transform the ER / EER model to a relational schema
  - Normalize the relations up to BCNF