**CSCI 403 DATABASE MANAGEMENT**

12.1 – Normalization

---

**Example Relation**

> Figure 1: One possible relation storing Mines course information:

<table>
<thead>
<tr>
<th>Instructor</th>
<th>Course ID</th>
<th>Section</th>
<th>Title</th>
<th>Office</th>
<th>Email</th>
</tr>
</thead>
<tbody>
<tr>
<td>Painter-Wakefield, Christopher</td>
<td>CSCI202</td>
<td>A</td>
<td>DATA STRUCTURES</td>
<td>BB 280I</td>
<td><a href="mailto:cpainter@mines.edu">cpainter@mines.edu</a></td>
</tr>
<tr>
<td>Mehta, Dinesh</td>
<td>CSCI406</td>
<td>A</td>
<td>ALGORITHMS</td>
<td>BB 280J</td>
<td><a href="mailto:dmehta@mines.edu">dmehta@mines.edu</a></td>
</tr>
<tr>
<td>Hellman, Keith</td>
<td>CSCI101</td>
<td>A</td>
<td>INTRO TO COMPUTER SCIENCE</td>
<td>BB 310F</td>
<td><a href="mailto:khellman@mines.edu">khellman@mines.edu</a></td>
</tr>
<tr>
<td>Hellman, Keith</td>
<td>CSCI101</td>
<td>B</td>
<td>INTRO TO COMPUTER SCIENCE</td>
<td>BB 310F</td>
<td><a href="mailto:khellman@mines.edu">khellman@mines.edu</a></td>
</tr>
<tr>
<td>Hellman, Keith</td>
<td>CSCI101</td>
<td>C</td>
<td>INTRO TO COMPUTER SCIENCE</td>
<td>BB 310F</td>
<td><a href="mailto:khellman@mines.edu">khellman@mines.edu</a></td>
</tr>
<tr>
<td>Hellman, Keith</td>
<td>CSCI274</td>
<td>A</td>
<td>INTRO TO LINUX OS</td>
<td>BB 310F</td>
<td><a href="mailto:khellman@mines.edu">khellman@mines.edu</a></td>
</tr>
</tbody>
</table>

---

**Functional Dependencies Review**

- Our primary tool for eliminating redundancy and modification anomalies
- A kind of constraint between two sets of attributes in a relation schema
- **Definition:** Given a relation schema $R$ and sets of attributes $X$ and $Y$, then we say a functional dependency $X \rightarrow Y$ exists if, whenever tuples $t_1$ and $t_2$ are two tuples from any relation $r(R)$ such that $t_1[X] = t_2[X]$, it is also true that $t_1[Y] = t_2[Y]$.
- **The lingo:** We say $X$ functionally determines $Y$, or $Y$ is functionally dependent on $X$.

---

**Functional Dependencies Review 2**

- In other words: if it is always true that whenever two tuples agree on attributes $X$, they also agree on $Y$, then $X \rightarrow Y$.
- **Example:** If we assert that an instructor is always associated with one office and email, then $\{\text{instructor} \} \rightarrow \{\text{office, email} \}$ is a functional dependency (FD) on the example table in figure 1.

---

**Normal Forms**

- Developed to define “good” design for a database
- Several forms: First normal form (1NF), Second (2NF), etc.
- Each normal form describes certain properties of a database
  - E.g., 1NF eliminates multivalued and compound attributes
  - Mostly later normal forms subsume earlier normal forms
- 1NF – 3NF are not terribly interesting stepping stones to the forms we care about:
  - Boyce-Codd Normal Form (BCNF)
- Fourth Normal Form (4NF)
- There exist even stronger normal forms (5NF etc.), but BCNF and 4NF suffice for most purposes.
Boyce-Codd Normal Form

Definition:
A relation \( R \) is in Boyce-Codd Normal Form (BCNF) if for every nontrivial functional dependency \( X \rightarrow A \) on \( R \), \( X \) is a superkey of \( R \).

BCNF Example

Consider our example relation schema in Figure 1:

One of the (non-trivial) functional dependencies we identified was:

\[ \text{instructor} \rightarrow \text{office} \]

Clearly, instructor is not a superkey of the relation.

Therefore, we say that the FD

\[ \text{instructor} \rightarrow \text{office} \]

violates BCNF, and the relation schema is not in BCNF.

Moving to BCNF

Our goal is a database in which every relation is in BCNF.

Fortunately, there is a straightforward algorithm for getting there.

- Find a relation schema not in BCNF
- Decompose it into two relation schemas, eliminating one of the BCNF violations
- Repeat
  (Details on next page)

Decomposition Algorithm

while some relation schema is not in BCNF:

- choose some relation schema \( R \) not in BCNF
- choose some FD \( X \rightarrow Y \) on \( R \) that violates BCNF
  (optional) expand \( Y \) so that \( Y = X^+ \) (closure of \( X \))
- let \( Z \) be all attributes of \( R \) not included in \( X \) or \( Y \)
- replace \( R \) with two new relations:
  \( R_1 \), containing attributes \( \{X, Y\} \)
  \( R_2 \), containing attributes \( \{X, Z\} \)

Decomposition Notes

- The final step above is accomplished simply by projection onto the attributes in \( R_2 \) and \( R_2 \). (Recall that this may result in fewer tuples.)
- After decomposing, you will need to establish which FDs now apply to \( R_1 \) and \( R_2 \), as well as determine their superkeys, in order to determine if they are now in BCNF.
- The optional step of expanding \( Y \) is recommended, as it tends to result in fewer, larger relation schemas, and may reduce the problem faster - e.g., consider decomposing on instructor \( 
\rightarrow \) office.

Decomposition Walkthrough

Let’s use the relation schema in Figure 1 as an example.

For this schema, we listed the following FDs:

- instructor \( 
\rightarrow \) office violates BCNF
- instructor \( 
\rightarrow \) email violates BCNF
- (course_id, section) \( 
\rightarrow \) instructor violates BCNF
- course_id \( 
\rightarrow \) title violates BCNF

What superkeys do we have?

Answer: any superset of our only key, which is (course_id, section).

Which FD’s violates BCNF?
Walkthrough 2

- Let’s pick our first violating FD to work with: instructor → office
- Next, expand the RHS as much as possible (we want the closure of instructor):
  instructor → (instructor, office, email)
- Now we decompose into two new tables, shown on the next slide:
  - R1 = π_instructor, office, email (R)
  - R2 = π_instructor, course_id, section, title (R)
- We can now discard the table from figure 1

Walkthrough 3

- Table R2 is now in BCNF ( yay!)
  - Note this was not guaranteed by the algorithm – we could have had other violating FDs
- Table R2 has a violating FD, though course_id → title

Walkthrough 4

Decompose R2 via course_id → title: course_id’ = (course_id, title)
Decompose into R3 and R4:
  - R3 = π_instructor, course_id (R2)
  - R4 = π_section, instructor, course_id (R2)

Walkthrough Wrap-up

- Done!
  - Three tables remain: R1, R3, R4
  - All non-essential/redundancy has been removed
  - Each table now represents a fundamental entity:
    - R1 = instructor info
    - R3 = course info
    - R4 = section info
- As a final note, this algorithm is not deterministic – you can different decompositions following different choices of FD to work with.

Next Time

- Correctness of decomposition algorithm
- Multi-valued dependencies and 4NF