These slides are based on “Database System Concepts” 6th edition book and are a modified version of the slides which accompany the book (http://codex.cs.yale.edu/avi/db-book/db6/slide-dir/index.html), in addition to the 2009/2012 CMSC 461 slides by Dr. Kalpakis
Logistics

- HW3 released today due 3/12/2018
- Phase 2 due Wednesday 3/7/2018
- Midterm 3/14/2018
Lecture Outline

• Relational Database Design
• First Normal Form (1NF)
• Functional Dependencies
• Boyce-Codd (BCNF)
• Third Normal Form (intro)
Lecture Outline

• *Relational Database Design*
  • First Normal Form (1NF)
  • Functional Dependencies
  • Boyce-Codd (BCNF)
  • Third Normal Form (intro)
Relational Database Design

- We want to move from the E-R diagram to a set of relations
  - Eliminate redundancy
  - Ensure design is complete
  - Ensure information is easily retrievable
- Going to learn about
  - normal form
  - functional dependencies
Design Alternatives: Combining Schemas

- Recall the instructor and department relations

<table>
<thead>
<tr>
<th>ID</th>
<th>name</th>
<th>dept_name</th>
<th>salary</th>
</tr>
</thead>
<tbody>
<tr>
<td>12121</td>
<td>Wu</td>
<td>Finance</td>
<td>90000.00</td>
</tr>
<tr>
<td>76766</td>
<td>Crick</td>
<td>Biology</td>
<td>72000.00</td>
</tr>
</tbody>
</table>

Instructor

<table>
<thead>
<tr>
<th>dept_name</th>
<th>building</th>
<th>budget</th>
</tr>
</thead>
<tbody>
<tr>
<td>Biology</td>
<td>Watson</td>
<td>90000.00</td>
</tr>
<tr>
<td>Comp. Sci.</td>
<td>Taylor</td>
<td>100000.00</td>
</tr>
<tr>
<td>Elec. Eng.</td>
<td>Taylor</td>
<td>85000.00</td>
</tr>
<tr>
<td>Finance</td>
<td>Painter</td>
<td>120000.00</td>
</tr>
<tr>
<td>History</td>
<td>Painter</td>
<td>50000.00</td>
</tr>
<tr>
<td>Music</td>
<td>Packard</td>
<td>80000.00</td>
</tr>
<tr>
<td>Physics</td>
<td>Watson</td>
<td>70000.00</td>
</tr>
</tbody>
</table>

Department
Design Alternatives: Combining Schemas

- Suppose we combine instructor and department into inst_dept
  - No connection to relationship set inst_dept
- Why is this a bad idea?

<table>
<thead>
<tr>
<th>ID</th>
<th>name</th>
<th>salary</th>
<th>dept_name</th>
<th>building</th>
<th>budget</th>
</tr>
</thead>
<tbody>
<tr>
<td>22222</td>
<td>Einstein</td>
<td>95000</td>
<td>Physics</td>
<td>Watson</td>
<td>70000</td>
</tr>
<tr>
<td>12121</td>
<td>Wu</td>
<td>90000</td>
<td>Finance</td>
<td>Painter</td>
<td>120000</td>
</tr>
<tr>
<td>32343</td>
<td>El Said</td>
<td>60000</td>
<td>History</td>
<td>Painter</td>
<td>50000</td>
</tr>
<tr>
<td>45565</td>
<td>Katz</td>
<td>75000</td>
<td>Comp. Sci.</td>
<td>Taylor</td>
<td>100000</td>
</tr>
<tr>
<td>98345</td>
<td>Kim</td>
<td>80000</td>
<td>Elec. Eng.</td>
<td>Taylor</td>
<td>85000</td>
</tr>
<tr>
<td>76766</td>
<td>Crick</td>
<td>72000</td>
<td>Biology</td>
<td>Watson</td>
<td>90000</td>
</tr>
<tr>
<td>10101</td>
<td>Srinivasan</td>
<td>65000</td>
<td>Comp. Sci.</td>
<td>Taylor</td>
<td>100000</td>
</tr>
<tr>
<td>58583</td>
<td>Califieri</td>
<td>62000</td>
<td>History</td>
<td>Painter</td>
<td>50000</td>
</tr>
<tr>
<td>83821</td>
<td>Brandt</td>
<td>92000</td>
<td>Comp. Sci.</td>
<td>Taylor</td>
<td>100000</td>
</tr>
<tr>
<td>15151</td>
<td>Mozart</td>
<td>40000</td>
<td>Music</td>
<td>Packard</td>
<td>80000</td>
</tr>
<tr>
<td>33456</td>
<td>Gold</td>
<td>87000</td>
<td>Physics</td>
<td>Watson</td>
<td>70000</td>
</tr>
<tr>
<td>76543</td>
<td>Singh</td>
<td>80000</td>
<td>Finance</td>
<td>Painter</td>
<td>120000</td>
</tr>
</tbody>
</table>

Based on and image from "Database System Concepts" book and slides, 6th edition
Design Alternatives: Combining Schemas

- Repetition of information

<table>
<thead>
<tr>
<th>ID</th>
<th>name</th>
<th>salary</th>
<th>dept_name</th>
<th>building</th>
<th>budget</th>
</tr>
</thead>
<tbody>
<tr>
<td>22222</td>
<td>Einstein</td>
<td>95000</td>
<td>Physics</td>
<td>Watson</td>
<td>70000</td>
</tr>
<tr>
<td>12121</td>
<td>Wu</td>
<td>90000</td>
<td>Finance</td>
<td>Painter</td>
<td>120000</td>
</tr>
<tr>
<td>32343</td>
<td>El Said</td>
<td>60000</td>
<td>History</td>
<td>Painter</td>
<td>50000</td>
</tr>
<tr>
<td>45565</td>
<td>Katz</td>
<td>75000</td>
<td>Comp. Sci.</td>
<td>Taylor</td>
<td>100000</td>
</tr>
<tr>
<td>98345</td>
<td>Kim</td>
<td>80000</td>
<td>Elec. Eng.</td>
<td>Taylor</td>
<td>85000</td>
</tr>
<tr>
<td>76766</td>
<td>Crick</td>
<td>72000</td>
<td>Biology</td>
<td>Watson</td>
<td>90000</td>
</tr>
<tr>
<td>10101</td>
<td>Srinivasan</td>
<td>65000</td>
<td>Comp. Sci.</td>
<td>Taylor</td>
<td>100000</td>
</tr>
<tr>
<td>58583</td>
<td>Califieri</td>
<td>62000</td>
<td>History</td>
<td>Painter</td>
<td>50000</td>
</tr>
<tr>
<td>83821</td>
<td>Brandt</td>
<td>92000</td>
<td>Comp. Sci.</td>
<td>Taylor</td>
<td>100000</td>
</tr>
<tr>
<td>15151</td>
<td>Mozart</td>
<td>40000</td>
<td>Music</td>
<td>Packard</td>
<td>80000</td>
</tr>
<tr>
<td>33456</td>
<td>Gold</td>
<td>87000</td>
<td>Physics</td>
<td>Watson</td>
<td>70000</td>
</tr>
<tr>
<td>76543</td>
<td>Singh</td>
<td>80000</td>
<td>Finance</td>
<td>Painter</td>
<td>120000</td>
</tr>
</tbody>
</table>

Based on and image from "Database System Concepts" book and slides, 6th edition
Combining Schemas without Repetition?

- Consider combining relations:
  
  \[ \text{sec\_class}(\text{sec\_id}, \text{building}, \text{room\_number}) \text{ and } \text{section}(\text{course\_id}, \text{sec\_id}, \text{semester}, \text{year}) \]

- into one relation
  
  \[ \text{section}(\text{course\_id}, \text{sec\_id}, \text{semester}, \text{year}, \text{building}, \text{room\_number}) \]

<table>
<thead>
<tr>
<th>sec_id</th>
<th>building</th>
<th>room_number</th>
<th>course_id</th>
<th>sec_id</th>
<th>semester</th>
<th>year</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Painter</td>
<td>514</td>
<td>BIO-101</td>
<td>1</td>
<td>Summer</td>
<td>2009</td>
</tr>
</tbody>
</table>

Based on and image from “Database System Concepts” book and slides, 6th edition
Design Alternatives: Smaller schemas

Given inst_dept was the result of our design.....

How would we know to split up (decompose) it into instructor and department?
Design Alternatives: Smaller schemas

- What can we say about this data?

<table>
<thead>
<tr>
<th>ID</th>
<th>name</th>
<th>salary</th>
<th>dept_name</th>
<th>building</th>
<th>budget</th>
</tr>
</thead>
<tbody>
<tr>
<td>22222</td>
<td>Einstein</td>
<td>95000</td>
<td>Physics</td>
<td>Watson</td>
<td>70000</td>
</tr>
<tr>
<td>12121</td>
<td>Wu</td>
<td>90000</td>
<td>Finance</td>
<td>Painter</td>
<td>120000</td>
</tr>
<tr>
<td>32343</td>
<td>El Said</td>
<td>60000</td>
<td>History</td>
<td>Painter</td>
<td>50000</td>
</tr>
<tr>
<td>45565</td>
<td>Katz</td>
<td>75000</td>
<td>Comp. Sci.</td>
<td>Taylor</td>
<td>100000</td>
</tr>
<tr>
<td>98345</td>
<td>Kim</td>
<td>80000</td>
<td>Elec. Eng.</td>
<td>Taylor</td>
<td>85000</td>
</tr>
<tr>
<td>76766</td>
<td>Crick</td>
<td>72000</td>
<td>Biology</td>
<td>Watson</td>
<td>90000</td>
</tr>
<tr>
<td>10101</td>
<td>Srinivasan</td>
<td>65000</td>
<td>Comp. Sci.</td>
<td>Taylor</td>
<td>100000</td>
</tr>
<tr>
<td>58583</td>
<td>Calieri</td>
<td>62000</td>
<td>History</td>
<td>Painter</td>
<td>50000</td>
</tr>
<tr>
<td>83821</td>
<td>Brandt</td>
<td>92000</td>
<td>Comp. Sci.</td>
<td>Taylor</td>
<td>100000</td>
</tr>
<tr>
<td>15151</td>
<td>Mozart</td>
<td>40000</td>
<td>Music</td>
<td>Packard</td>
<td>80000</td>
</tr>
<tr>
<td>33456</td>
<td>Gold</td>
<td>87000</td>
<td>Physics</td>
<td>Watson</td>
<td>70000</td>
</tr>
<tr>
<td>76543</td>
<td>Singh</td>
<td>80000</td>
<td>Finance</td>
<td>Painter</td>
<td>120000</td>
</tr>
</tbody>
</table>

Based on and image from "Database System Concepts" book and slides, 6th edition
Design Alternatives: Smaller schemas

- Write a rule:

  “if there were a schema \((\text{dept\_name}, \text{building}, \text{budget})\), then \text{dept\_name} would be able to serve as the primary key”
Design Alternatives: Smaller schemas

- Denote as a \textit{functional dependency}:
  \[
  \text{dept\_name} \rightarrow \text{building, budget}
  \]

- In \textit{inst\_dept}, because \textit{dept\_name} is not a primary key, the building and budget of a department may have to be repeated
  - This indicates the need to decompose \textit{inst\_dept}
Design Alternatives: Smaller schemas

Not all decompositions are good.

Suppose we decompose:

\( \text{employee}(ID, \text{name, street, city, salary}) \)

into

\( \text{employee1 (ID, name)} \)
\( \text{employee2 (name, street, city, salary)} \)
A Lossy Decomposition

Based on and image from “Database System Concepts” book and slides, 6th edition
Lossless Join Decomposition

Decomposition of $R = (A, B, C)$

$R_1 = (A, B)$ \quad $R_2 = (B, C)$

$\Pi_{A,B}(r) \Join \Pi_{B,C}(r)$

Based on and image from “Database System Concepts” book and slides, 6th edition
Another Example

- Consider the relation schema:
  - \textit{Lending-schema} = (\textit{branchName}, \textit{branchCity}, \textit{assets}, \textit{customerName}, \textit{loanNumber}, \textit{amount})

<table>
<thead>
<tr>
<th>branch-name</th>
<th>branch-city</th>
<th>assets</th>
<th>customer-name</th>
<th>loan-number</th>
<th>amount</th>
</tr>
</thead>
<tbody>
<tr>
<td>Downtown</td>
<td>Brooklyn</td>
<td>9000000</td>
<td>Jones</td>
<td>L-17</td>
<td>1000</td>
</tr>
<tr>
<td>Redwood</td>
<td>Palo Alto</td>
<td>2100000</td>
<td>Smith</td>
<td>L-23</td>
<td>2000</td>
</tr>
<tr>
<td>Perryridge</td>
<td>Horseneck</td>
<td>1700000</td>
<td>Hayes</td>
<td>L-15</td>
<td>1500</td>
</tr>
<tr>
<td>Downtown</td>
<td>Brooklyn</td>
<td>9000000</td>
<td>Jackson</td>
<td>L-14</td>
<td>1500</td>
</tr>
</tbody>
</table>

Based on and image from “Database System Concepts” book and slides, 6th edition
Another Example

- Redundancy:
  - Data for branchName, branchCity, assets are repeated for each loan that a branch makes
  - Wastes space
  - Complicates updating, introducing possibility of inconsistency of assets value

- Null values
  - Cannot store information about a branch if no loans exist
  - Can use null values, but they are difficult to handle.
Decompose the relation schema Lending-schema into:

\[
\text{BranchSchema} = (\text{branchName}, \text{branchCity}, \text{assets})
\]

\[
\text{LoanSchema} = (\text{customerName}, \text{loanNumber}, \text{branchName}, \text{amount})
\]

All attributes of an original schema (R) must appear in the decomposition (R₁, R₂):

\[
R = R₁ \cup R₂
\]

**Lossless-join** decomposition. For all possible relations \( r \) on schema \( R \)

\[
r = \prod_{R₁} (r) \Join \prod_{R₂} (r)
\]

Based on and image from "Database System Concepts" book and slides, 6th edition
Lecture Outline

• Relational Database Design
• *First Normal Form (1NF)*
• Functional Dependencies
• Boyce-Codd (BCNF)
• Third Normal Form
First Normal Form (1NF)

- In the relational model attributes have no substructure
  - composite attributes - each component becomes its own attribute
  - multivalued attributes - one tuple for each item
- Domain is *atomic* if its elements are considered to be indivisible units
  - Examples of non-atomic domains:
    - Set of names, composite attributes
    - Identification numbers like CS001 where department is combined with employee number
    - Fine line: ‘CS101’ might be a course identifier and could be interpreted as atomic
First Normal Form (1NF)

- Non-atomic values complicate storage and encourage redundant (repeated) storage of data
  - Example: Set of accounts stored with each customer, and set of owners stored with each account
First Normal Form (1NF)

- A relational schema $R$ is in \textit{first normal form} if the domains of all attributes of $R$ are atomic.
Atomicity is actually a property of how the elements of the domain are used

- Example: Strings would normally be considered indivisible
- Suppose that students are given roll numbers which are strings of the form CS0012 or EE1127
- If the first two characters are extracted to find the department, the domain of roll numbers is not atomic.
- Doing so is a bad idea: leads to encoding of information in application program rather than in the database
Is it atomic?

- Address
- Course ID (CS-101)
- Student Name
- SSN
- ISBN Number
Convert it to 1NF

<table>
<thead>
<tr>
<th>ID</th>
<th>Color</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Blue,Red</td>
</tr>
<tr>
<td>2</td>
<td>Yellow,Brown</td>
</tr>
<tr>
<td>3</td>
<td>Orange,Green</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>ID</th>
<th>Name</th>
<th>Address</th>
<th>Order</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>John Doe</td>
<td>13101 Brandley Lane</td>
<td>Cell Phone</td>
</tr>
<tr>
<td>2</td>
<td>Jackie Click</td>
<td>4531 Tinker Road</td>
<td>Charger</td>
</tr>
<tr>
<td>3</td>
<td>Brad Dunkin</td>
<td>8593 Gerwin Avenue</td>
<td>Cell Phone Case</td>
</tr>
</tbody>
</table>
Relational database design requires that we find a “good” collection of relation schemas.

A bad design may lead to:
- Repetition of Information
- Inability to represent certain information

Design Goals:
- Avoid redundant data
- Ensure that relationships among attributes are represented
- Facilitate the checking of updates for violation of database integrity constraints
Goal: Devise a Theory for the following:

- Decide whether a particular relation R is in “good” form
- In the case that a relation R is not in “good” form, decompose it into a set of relations \{R_1, R_2, ..., R_n\} such that
  - each relation is in good form
  - the decomposition is a lossless-join decomposition
- This theory is based on:
  - functional dependencies
  - multivalued dependencies

Based on and image from “Database System Concepts” book and slides, 6th edition
Lecture Outline

• Relational Database Design
• First Normal Form (1NF)
• **Functional Dependencies**
• Boyce-Codd (BCNF)
• Third Normal Form
Functional Dependencies

- Constraints on the set of legal relations
- Require that the value for a certain set of attributes determines uniquely the value for another set of attributes
- A functional dependency is a generalization of the notion of a key
Remember

- $r$ is a relation
- $r(R)$ is the schema for the relation $r$
- $R$ denotes the set of attributes
- $K$ represents the set of attributes that is the superkey
- A superkey – set of attributes that uniquely identify a tuple
Functional Dependencies

Let $R$ be a relation schema

$\alpha \subseteq R$ and $\beta \subseteq R$

The functional dependency $\alpha \rightarrow \beta$

holds on $R$ if and only if for any legal relations $r(R)$, whenever any two tuples $t_1$ and $t_2$ of $r$ agree on the attributes $\alpha$, they also agree on the attributes $\beta$. That is

$t_1[\alpha] = t_2[\alpha] \Rightarrow t_1[\beta] = t_2[\beta]$

Example: Consider $r(A,B)$ with the following instance of $r$

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>4</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>5</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>7</td>
<td></td>
</tr>
</tbody>
</table>

On this instance, $A \rightarrow B$ does NOT hold, but $B \rightarrow A$ does hold
Functional Dependencies

- Functional dependencies allow us to express constraints that cannot be expressed using superkeys. Consider the schema:

  \[\text{inst_dept (ID, name, salary, dept\_name, building, budget)}\]

We expect these functional dependencies to hold:

\[\text{dept\_name} \rightarrow \text{building}\]

but would not expect the following to hold:

\[\text{dept\_name} \rightarrow \text{salary}\]
We use functional dependencies to:

- test relations to see if they are legal under a given set of functional dependencies
  - If a relation \( r \) is legal under a set \( F \) of functional dependencies, we say that \( r \) satisfies \( F \)
- specify constraints on the set of legal relations

We say that \( F \) holds on \( R \) if all legal relations on \( R \) satisfy the set of functional dependencies \( F \)

Note: A specific instance of a relation schema may satisfy a functional dependency even if the functional dependency does not hold on all legal instances

- For example, a specific instance of instructor may sometimes satisfy
  
  \[ \text{name} \rightarrow \text{ID} \]
Functional Dependencies

- A functional dependency is trivial if it is satisfied by all instances of a relation
  - Example:
    - $ID, name \rightarrow ID$
    - $name \rightarrow name$
  - In general, $\alpha \rightarrow \beta$ is trivial if $\beta \subseteq \alpha$
Functional Dependencies
Examples

Assume schema:

\textit{student}(\textit{student\_id, first\_name, last\_name, major, SSN})

Which are true in regards to functional dependencies:

\texttt{student\_id → last\_name} \\
\texttt{last\_name → student\_id} \\
\texttt{student\_id → last\_name, major, SSN, student\_id} \\
\texttt{SSN → student\_id, last\_name, major, SSN} \\
\texttt{first\_name→ last\_name} \\
\texttt{last\_name → last\_name}
Functional Dependencies
Examples

Assume schema:

\[
\text{student}(\text{student_id}, \text{first_name}, \text{last_name}, \text{major}, \text{SSN})
\]

Which are true in regards to functional dependencies:

\[
\begin{align*}
\text{student_id} \rightarrow \text{last_name} & \quad \text{TRUE} \\
\text{last_name} \rightarrow \text{student_id} & \quad \text{FALSE} \\
\text{student_id} \rightarrow \text{last_name}, \text{major}, \text{SSN}, \text{student_id} & \quad \text{TRUE} \\
\text{SSN} \rightarrow \text{student_id}, \text{last_name}, \text{major}, \text{SSN} & \quad \text{TRUE} \\
\text{first_name} \rightarrow \text{last_name} & \quad \text{FALSE} \\
\text{last_name} \rightarrow \text{last_name} & \quad \text{TRUE}
\end{align*}
\]
Closure of a Set of Functional Dependencies

- Given a set $F$ of functional dependencies, there are certain other functional dependencies that are logically implied by $F$
  - For example:
    Given a schema $r(A,B,C)$
    If $A \rightarrow B$ and $B \rightarrow C$
    then we can infer that $A \rightarrow C$
- The set of all functional dependencies logically implied by $F$ is the closure of $F$
- We denote the closure of $F$ by $F^+$
- $F^+$ is a superset of $F$
Lecture Outline

• Relational Database Design
• First Normal Form (1NF)
• Functional Dependencies
  • **Boyce-Codd (BCNF)**
• Third Normal Form (Intro)
Boyce-Codd Normal Form

A relation schema $R$ is in BCNF with respect to a set $F$ of functional dependencies if for all functional dependencies in $F^+$ of the form

$$\alpha \rightarrow \beta$$

where $\alpha \subseteq R$ and $\beta \subseteq R$, at least one of the following holds:

- $\alpha \rightarrow \beta$ is trivial (i.e., $\beta \subseteq \alpha$)
- $\alpha$ is a superkey for $R$

Example schema not in BCNF:

```
instr_dept (ID, name, salary, dept_name, building, budget )
```

because $dept\_name \rightarrow building, budget$ holds on $instr\_dept$, but $dept\_name$ is not a superkey
Boyce-Codd Normal Form

Are these schemas in BCNF:

instructor (ID, name, dept_name, salary)
$ID \rightarrow name, dept_name, salary$

department(dept_name, building, budget)
$dept_name \rightarrow building, budget$

Why or why not?
Boyce-Codd Normal Form

Are these schemas in BCNF:

instructor (ID, name, dept_name, salary)
ID → name, dept_name, salary

department(dept_name, building, budget)
dept_name → building, budget

YES – ID is superkey

YES – dept_name is superkey
Decomposing a Schema into BCNF

- Suppose we have a schema $R$ and a non-trivial dependency $\alpha \rightarrow \beta$ causes a violation of BCNF. We decompose $R$ into:

  \[(\alpha \cup \beta)\]

  \[(R - (\beta - \alpha))\]

- In our example:

  \[\text{instr_dept}(\text{ID, name, salary, dept_name, building, budget})\]

  $\alpha = \text{dept_name}$

  $\beta = \text{building, budget}$

  and \text{inst_dept} is replaced by

  \[(\alpha \cup \beta) = (\text{dept_name, building, budget})\]

  \[(R - (\beta - \alpha)) = (\text{ID, name, salary, dept_name})\]
Convert it to BCNF

Schema:

Student(ID, Name, AdvisorID, AdvisorName)

What are the functional dependencies?
Convert it to BCNF

Schema:

Student(ID, Name, AdvisorID, AdvisorName)

What are the functional dependencies?

ID -> Name
AdvisorID -> AdvisorName

What uniquely identifies the tuples?
Convert it to BCNF

Schema:

Student(ID, Name, AdvisorID, AdvisorName)

What are the functional dependencies?

ID -> Name
AdvisorID -> AdvisorName

What uniquely identifies the tuples?
(ID, AdvisorID)

Is there a BCNF violation?
Convert it to BCNF

Schema:

Student(ID, Name, AdvisorID, AdvisorName)

What are the functional dependencies?

ID -> Name
AdvisorID -> AdvisorName

What is the primary key?
(ID, AdvisorID)

Is there a BCNF violation? YES!
Convert it to BCNF

Schema:
Student(ID,Name,AdvisorID,AdvisorName)

What are the functional dependencies?
ID -> Name
AdvisorID -> AdvisorName

What is the primary key?
(ID,AdvisorID)

Is there a BCNF violation? YES!

Use ID-> Name to decompose R- (Name-ID)= (ID,AdvisorID,AdvisorName) and ID union Name = (ID,Name)
BCNF and Dependency Preservation

- Constraints, including functional dependencies, are costly to check in practice unless they pertain to only one relation.
- If it is sufficient to test only those dependencies on each individual relation of a decomposition in order to ensure that all functional dependencies hold, then that decomposition is dependency preserving.
- Because it is not always possible to achieve both BCNF and dependency preservation, we consider a weaker normal form, known as third normal form.
Lecture Outline

- Relational Database Design
- First Normal Form (1NF)
- Functional Dependencies
- Boyce-Codd (BCNF)
  - Third Normal Form (Intro)
Third Normal Form

- A relation schema $R$ is in **third normal form (3NF)** if for all:
  \[ \alpha \rightarrow \beta \text{ in } F^+ \]
  at least one of the following holds:
  - $\alpha \rightarrow \beta$ is trivial
  - $\alpha$ is a superkey for $R$
  - Each attribute $A$ in $\beta - \alpha$ is contained in a candidate key for $R$.
  
  (NOTE: each attribute may be in a different candidate key)

- If a relation is in BCNF it is in 3NF (since in BCNF one of the first two conditions above must hold).

- Third condition is a minimal relaxation of BCNF to ensure dependency preservation (will see why later).