Lecture 12 - Chapter 8 Relational Database Design Part 2

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

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

• Midterm Review
• Normalization
• Boyce-Codd (BCNF)
• Third Normal Form
• Functional Dependency Theory
Lecture Outline

• **Midterm Review**
• Normalization
• Boyce-Codd (BCNF)
• Third Normal Form
• Functional Dependency Theory
Midterm

- See study guide

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

- Midterm Review
- **Normalization**
- Boyce-Codd (BCNF)
- Third Normal Form
- Functional Dependency Theory
Why Normalize?
Why Normalize?

- Reduce the amount of duplicate data
- Reduce data modification issues
- Simplify queries

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

- First (we will cover a lot)
- Second (we will briefly cover)
- Third and BCNF (we will cover a lot)
- Fourth (we will briefly cover)
- Fifth (we will not cover)
First Normal Form

- Attributes contains atomic values
- Eliminate composite and multi-valued attributes

Second Normal Form

- If each attribute in $R$ meets one of the following:
  - It appears in a candidate key
  - It is not partially dependent on a candidate key

Therefore, if $R$ is in 1st normal form and its non-key attributes are functionally dependent on the candidate key it is in second normal form.
Second Normal Form

Entity Integrity Violation: P# is a part of primary key!

Third Normal Form

- If in 2nd normal form and
- Contains only attributes dependent on the primary key and not other attributes

<table>
<thead>
<tr>
<th>Customer</th>
<th>C#</th>
<th>Cname</th>
<th>Ccity</th>
<th>Cphone</th>
<th>Salesperson</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td></td>
<td>Codd</td>
<td>London</td>
<td>2263035</td>
<td>Smith</td>
</tr>
<tr>
<td>2</td>
<td></td>
<td>Martin</td>
<td>Paris</td>
<td>5555910</td>
<td>Ducruer</td>
</tr>
<tr>
<td>3</td>
<td></td>
<td>Deen</td>
<td>London</td>
<td>2234391</td>
<td>Smith</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Sarawak</td>
<td></td>
<td></td>
<td>Fatimah</td>
</tr>
</tbody>
</table>

C# -> Cname, Ccity, Cphone, Salesperson
Salesperson has indirect dependency
Boyce-Codd Normal Form

- Remember BCNF is stricter than 3NF
- So if it is BCNF, then it is 3NF
- However if it is 3NF, it may not be BCNF

<table>
<thead>
<tr>
<th>Student</th>
<th>Course</th>
<th>Teacher</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sok</td>
<td>DB</td>
<td>John</td>
</tr>
<tr>
<td>Sao</td>
<td>DB</td>
<td>William</td>
</tr>
<tr>
<td>Chan</td>
<td>E-Commerce</td>
<td>Todd</td>
</tr>
<tr>
<td>Sok</td>
<td>E-Commerce</td>
<td>Todd</td>
</tr>
<tr>
<td>Chan</td>
<td>DB</td>
<td>William</td>
</tr>
</tbody>
</table>

- Key: \{Student, Course\}
- Functional Dependency:
  - \{Student, Course\} → Teacher
  - Teacher → Course
- Problem: Teacher is not a superkey but determines Course.
Fourth Normal Form

- Has to be in BCNF
- Requires understanding multivalued dependencies
- Given an entity, should not contain 2 or more independent multi-valued facts

<table>
<thead>
<tr>
<th>Employee ID</th>
<th>Language</th>
<th>Operating System</th>
</tr>
</thead>
<tbody>
<tr>
<td>1212</td>
<td>C++</td>
<td>Windows</td>
</tr>
<tr>
<td>1212</td>
<td>Java</td>
<td>Windows</td>
</tr>
<tr>
<td>1212</td>
<td>Python</td>
<td>Windows</td>
</tr>
<tr>
<td>1212</td>
<td>Python</td>
<td>Linux</td>
</tr>
<tr>
<td>1212</td>
<td>Java</td>
<td>Linux</td>
</tr>
</tbody>
</table>

Based on and image from “Database System Concepts” book and slides, 6th edition
## Fourth Normal Form

- Has to be in BCNF
- Requires understanding multivalued dependencies
- Given an entity, should not contain 2 or more independent multi-valued facts

<table>
<thead>
<tr>
<th>Employee ID</th>
<th>Language</th>
</tr>
</thead>
<tbody>
<tr>
<td>1212</td>
<td>C++</td>
</tr>
<tr>
<td>1212</td>
<td>Java</td>
</tr>
<tr>
<td>1212</td>
<td>Python</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Employee ID</th>
<th>Operating System</th>
</tr>
</thead>
<tbody>
<tr>
<td>1212</td>
<td>Windows</td>
</tr>
<tr>
<td>1212</td>
<td>Linux</td>
</tr>
<tr>
<td>1212</td>
<td>Linux</td>
</tr>
</tbody>
</table>
Fourth Normal Form

Another example (more on 4NF later)

<table>
<thead>
<tr>
<th>Car_Model (PK)</th>
<th>Engine_Type (PK)</th>
<th>Color (PK)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mustang</td>
<td>3.7L V6</td>
<td>Red</td>
</tr>
<tr>
<td>Mustang</td>
<td>3.7L V6</td>
<td>Blue</td>
</tr>
<tr>
<td>Mustang</td>
<td>5.0L V8</td>
<td>Red</td>
</tr>
<tr>
<td>Taurus</td>
<td>3.5L V6</td>
<td>Green</td>
</tr>
<tr>
<td>Taurus</td>
<td>2.0L Eco</td>
<td>Green</td>
</tr>
</tbody>
</table>

Based on and image from "Database System Concepts" book and slides, 6th edition
Simplified: The Normal Forms

A nice simple discussion of normal forms (not 100% precise, but close enough)


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

- Midterm Review
- Normalization
- Boyce-Codd (BCNF)
- Third Normal Form
- Functional Dependency Theory
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 → 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:

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

  $\alpha = \text{dept_name}$

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

  and $\text{inst_dept}$ is replaced by

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

  $(R - (\beta - \alpha)) = (\text{ID, name, salary, dept_name})$

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

Based on and image from “Database System Concepts” book and slides, 6th edition
Convert it to BCNF

Schema:

\textbf{Student(}ID,Name,AdvisorID,AdvisorName)\textbf{)}

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
(ID,AdvisorID,AdvisorName) and (ID,Name)
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

• Midterm Review
• Normalization
• Boyce-Codd (BCNF)
• *Third Normal Form*
• Functional Dependency Theory
Third Normal Form

- A relation schema $R$ is in **third normal form (3NF)** if for all:
  
  $\alpha \rightarrow \beta$ 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).
Third Normal Form

- Given `dept_advisor` with dependencies:
  - `i_ID → dept_name`
  - `s_ID, dept_name → i_ID`

- `i_ID → dept_name` make `dept_advisor` not BCNF
  - \( \alpha = i_ID \)
  - \( \beta = \text{dept_name} \)
  - \( \beta \sim \alpha = \text{dept_name} \)

- But since `s_ID, dept_name → i_ID` holds on `dept_advisor` then `dept_name` is a candidate key which means

- `dept_advisor` is in 3NF
Goals of Normalization

- Let R be a relation scheme with a set F of functional dependencies
- Decide whether a relation scheme R is in “good” form
- In the case that a relation scheme R is not in “good” form, decompose it into a set of relation schemes \{R_1, R_2, ..., R_n\} such that:
  - each relation scheme is in good form
  - the decomposition is a lossless-join decomposition
  - Preferably, the decomposition should be dependency preserving
How Good is BCNF?

- There are database schemas in BCNF that do not seem to be sufficiently normalized
- Consider a relation inst_info (ID, child_name, phone)
  - where an instructor may have more than one phone and can have multiple children

<table>
<thead>
<tr>
<th>ID</th>
<th>child_name</th>
<th>phone</th>
</tr>
</thead>
<tbody>
<tr>
<td>99999</td>
<td>David</td>
<td>512-555-1234</td>
</tr>
<tr>
<td>99999</td>
<td>David</td>
<td>512-555-4321</td>
</tr>
<tr>
<td>99999</td>
<td>William</td>
<td>512-555-1234</td>
</tr>
<tr>
<td>99999</td>
<td>Willian</td>
<td>512-555-4321</td>
</tr>
</tbody>
</table>
How Good is BCNF?

- There are no non-trivial functional dependencies and therefore the relation is in BCNF
- Insertion anomalies – i.e., if we add a phone 981-992-3443 to 99999, we need to add two tuples (99999, David, 981-992-3443) (99999, William, 981-992-3443)
How Good is BCNF?

- Therefore, it is better to decompose inst_info into:

<table>
<thead>
<tr>
<th>ID</th>
<th>child_name</th>
</tr>
</thead>
<tbody>
<tr>
<td>99999</td>
<td>David</td>
</tr>
<tr>
<td>99999</td>
<td>David</td>
</tr>
<tr>
<td>99999</td>
<td>William</td>
</tr>
<tr>
<td>99999</td>
<td>Willian</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>ID</th>
<th>phone</th>
</tr>
</thead>
<tbody>
<tr>
<td>99999</td>
<td>512-555-1234</td>
</tr>
<tr>
<td>99999</td>
<td>512-555-4321</td>
</tr>
<tr>
<td>99999</td>
<td>512-555-1234</td>
</tr>
<tr>
<td>99999</td>
<td>512-555-4321</td>
</tr>
</tbody>
</table>

This suggests the need for higher normal forms, such as Fourth Normal Form (4NF), which we shall see later.
Lecture Outline

• Midterm Review
• Normalization
• Boyce-Codd (BCNF)
• Third Normal Form
• Functional Dependency Theory
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 $r$

  $\begin{array}{cc}
  1 & 4 \\
  1 & 5 \\
  3 & 7 \\
  \end{array}$

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

Based on and image from "Database System Concepts" book and slides, 6th edition
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 $\beta \subseteq \alpha$
Assume schema:

student(student_id, first_name, last_name, major, SSN)

Which are true in regards to functional dependencies:

- student_id → last_name: TRUE
- last_name → student_id: FALSE
- student_id → last_name, major, SSN, student_id: TRUE
- SSN → student_id, last_name, major, SSN: TRUE
- first_name → last_name: FALSE
- last_name → last_name: TRUE
We now consider the formal theory that tells us which functional dependencies are implied logically by a given set of functional dependencies.
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 $\text{r}(A,B,C)$
      - 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 $F$.

Based on and image from "Database System Concepts" book and slides, 6th edition.
Closure of a Set of Functional Dependencies

• We can find $F^+$, the closure of $F$, by repeatedly applying

  **Armstrong’s Axioms:**
  
  - if $\beta \subseteq \alpha$, then $\alpha \rightarrow \beta$  \hspace{1cm} (reflexivity)
  - if $\alpha \rightarrow \beta$, then $\gamma \alpha \rightarrow \gamma \beta$  \hspace{1cm} (augmentation)
  - if $\alpha \rightarrow \beta$, and $\beta \rightarrow \gamma$, then $\alpha \rightarrow \gamma$  \hspace{1cm} (transitivity)

• These rules are
  
  - **sound** (generate only functional dependencies that actually hold), and
  
  - **complete** (generate all functional dependencies that hold).
Closure of a set of Functional Dependencies

- Additional rules:
  - If $\alpha \rightarrow \beta$ holds and $\alpha \rightarrow \gamma$ holds, then $\alpha \rightarrow \beta \gamma$ holds (union)
  - If $\alpha \rightarrow \beta \gamma$ holds, then $\alpha \rightarrow \beta$ holds and $\alpha \rightarrow \gamma$ holds (decomposition)
  - If $\alpha \rightarrow \beta$ holds and $\gamma \beta \rightarrow \delta$ holds, then $\alpha \gamma \rightarrow \delta$ holds (pseudotransitivity)

The above rules can be inferred from Armstrong’s axioms.
Closure of a set of Functional Dependencies Example

- \( R = (A, B, C, G, H, I) \)
- \( F = \{ A \rightarrow B, A \rightarrow C, CG \rightarrow H, CG \rightarrow I, B \rightarrow H \} \)

- some members of \( F^+ \)
  - \( A \rightarrow H \)
    - by transitivity from \( A \rightarrow B \) and \( B \rightarrow H \)
  - \( AG \rightarrow I \)
    - by augmenting \( A \rightarrow C \) with \( G \), to get \( AG \rightarrow CG \) and then transitivity with \( CG \rightarrow I \)
  - \( CG \rightarrow HI \)
    - by union rule, since \( CG \rightarrow H \) and \( CG \rightarrow I \), implies \( CG \rightarrow HI \)

Based on and image from “Database System Concepts” book and slides, 6th edition
Computing $F^+$

- To compute the closure of a set of functional dependencies $F$:

$$F^+ = F$$

repeat
for each functional dependency $f$ in $F^+$
    apply reflexivity and augmentation rules on $f$
    add the resulting functional dependencies to $F^+$
for each pair of functional dependencies $f_1$ and $f_2$ in $F^+$
    if $f_1$ and $f_2$ can be combined using transitivity
    then add the resulting functional dependency to $F^+$
until $F^+$ does not change any further

NOTE: We shall see an alternative procedure for this task later