CMSC424: Database Design
Introduction/Overview

Professor: Pete Keleher
keleher@cs.umd.edu

Outline

- Mechanisms and definitions to work with FDs
  - Closures, candidate keys, canonical covers etc…
  - Armstrong axioms
- Decompositions
  - Loss-less decompositions, Dependency-preserving decompositions
- BCNF
  - How to achieve a BCNF schema
- BCNF may not preserve dependencies
- 3NF: Solves the above problem
- BCNF allows for redundancy
- 4NF: Solves the above problem
Approach

1. We will encode and list all our knowledge about the schema
   ◦ Functional dependencies (FDs)
     ◦ SSN → name (means: SSN “implies” name)
     ◦ If two tuples have the same “SSN”, they must have the same “name”
     ◦ movietitle → length ❏ Not true.
     ◦ But, (movietitle, movieYear) → length --- Probably.
2. We will define a set of rules that the schema must follow to be considered good
   ◦ “Normal forms”: 1NF, 2NF, 3NF, BCNF, 4NF, …
   ◦ A normal form specifies constraints on the schemas and FDs
3. If not in a “normal form”, we modify the schema

FDs: Example

<table>
<thead>
<tr>
<th>Course ID</th>
<th>Course Name</th>
<th>Dept Name</th>
<th>Credits</th>
<th>Section ID</th>
<th>Semester</th>
<th>Year</th>
<th>Building</th>
<th>Room No.</th>
<th>Capacity</th>
<th>Time Slot ID</th>
</tr>
</thead>
</table>

Functional dependencies

- course_id → course_name, dept_name, credits
- building, room_number → capacity
- course_id, section_id, semester, year → building, room_number, time_slot_id
Functional Dependencies

Let \( r(R) \) be a relation schema and
\[ \alpha \subseteq R \quad \text{and} \quad \beta \subseteq R \]
The functional dependency
\[ \alpha \rightarrow \beta \]
holds on \( R \) iff for any legal relations \( r(R) \), whenever two tuples \( t_1 \) and \( t_2 \) of \( r \) have same values for \( \alpha \), they have same values for \( \beta \).
\[ t_1[\alpha] = t_2[\alpha] \implies t_1[\beta] = t_2[\beta] \]

Example:

\[
\begin{array}{cc}
A & B \\
1 & 4 \\
1 & 5 \\
3 & 7 \\
\end{array}
\]

In this instance, \( A \rightarrow B \) does NOT hold, but \( B \rightarrow A \) does hold.

---

### Functional Dependencies

- Difference between holding on an instance and holding on all legal relations.

<table>
<thead>
<tr>
<th>Title</th>
<th>Year</th>
<th>Length</th>
<th>inColor</th>
<th>StudioName</th>
<th>prodC#</th>
<th>StarName</th>
</tr>
</thead>
<tbody>
<tr>
<td>Star wars</td>
<td>1977</td>
<td>121</td>
<td>Yes</td>
<td>Fox</td>
<td>128</td>
<td>Hamill</td>
</tr>
<tr>
<td>Star wars</td>
<td>1977</td>
<td>121</td>
<td>Yes</td>
<td>Fox</td>
<td>128</td>
<td>Fisher</td>
</tr>
<tr>
<td>Star wars</td>
<td>1977</td>
<td>121</td>
<td>Yes</td>
<td>Fox</td>
<td>128</td>
<td>H. Ford</td>
</tr>
<tr>
<td>King Kong</td>
<td>1933</td>
<td>100</td>
<td>no</td>
<td>RKO</td>
<td>20</td>
<td>Fay</td>
</tr>
</tbody>
</table>

- \( Title \rightarrow Year \) holds on this instance.

- Is this a true functional dependency?
  - No. Two movies in different years can have the same name.
- Can’t draw conclusions based on a single instance
  - Need domain knowledge to decide which FDs hold.

---

An instance is the value of a \( r \) at a particular point in time.
Functional Dependencies

- Functional dependencies and *keys*
  - A *key* constraint is a specific form of a FD.
  - E.g. if $\alpha$ is a superkey for $R$, then:
    \[ \alpha \to R \]
  - Similarly for *candidate keys and primary keys*.

- Deriving FDs
  - A set of FDs may imply other FDs
  - E.g. If $A \to B$, and $B \to C$, then clearly $A \to C$
  - *We will see a formal method for inferring this later*

Definitions

1. A relation instance $r$ *satisfies* a set of functional dependencies, $F$, if the FDs hold on the relation

2. $F$ holds on a relation schema $R$ if no legal (allowable) relation instance of $R$ violates it

3. A functional dependency, $\alpha \to \beta$, is called *trivial* if:
   - $\beta$ is a subset of $\alpha$
   - E.g. Movieyear, length $\to$ length

4. Given a set of functional dependencies, $F$, its *closure*, $F^+$, is all the FDs that are implied by FDs in $F$. 
Approach

1. We will encode and list all our knowledge about the schema
   - Functional dependencies (FDs)
   - Also:
     - Multi-valued dependencies (briefly discuss later)
     - Join dependencies etc…

2. We will define a set of rules that the schema must follow to be considered good
   - “Normal forms”: 1NF, 2NF, 3NF, BCNF, 4NF, …
   - A normal form specifies constraints on the schemas and FDs

3. If not in a “normal form”, we modify the schema

BCNF: Boyce-Codd Normal Form

A relation schema \( R \) is “in BCNF” if:

- Every functional dependency \( \alpha \rightarrow \beta \) that holds on it is EITHER:
  1. Trivial OR
  2. \( \alpha \) is a superkey of \( R \)

Why is BCNF good?

- Guarantees that there can be no redundancy because of a functional dependency
- Consider a relation \( r(A, B, C, D) \) with functional dependency
  - \( A \rightarrow B \) and two tuples: \((a1, b1, c1, d1)\), and \((a1, b1, c2, d2)\)
  - \( b1 \) is repeated because of the functional dependency
  - BUT this relation is not in BCNF
  - \( A \rightarrow B \) is neither trivial nor is \( A \) a superkey for the relation
BCNF and Redundancy

Why does redundancy arise?
- Given a FD, \( \alpha \rightarrow \beta \), if \( \alpha \) is repeated (\( \beta - \alpha \)) has to be repeated
  1. If rule 1 is satisfied, (\( \beta - \alpha \)) is empty, so not a problem.
  2. If rule 2 is satisfied, then \( \alpha \) can’t be repeated, so this doesn’t happen either

- Hence no redundancy because of FDs
  - Redundancy may exist because of other types of dependencies
    - Higher normal forms used for that (specifically, 4NF)
  - Data may naturally have duplicated/redundant data
    - We can’t control that unless a FD or some other dependency is defined

Approach

1. We will encode and list all our knowledge about the schema
   - Functional dependencies (FDs); Multi-valued dependencies; Join dependencies etc…

2. We will define a set of rules that the schema must follow to be considered good
   - “Normal forms”: 1NF, 2NF, 3NF, BCNF, 4NF, …
   - A normal form specifies constraints on the schemas and FDs

3. If not in a “normal form”, we modify the schema
   - Through lossless decomposition (splitting)
   - Or direct construction using the dependencies information
What if the schema is not in BCNF?
- Decompose (split) the schema into two pieces.

From the previous example: split the schema into:
- \( r_1(A, B), \ r_2(A, C, D) \)
- The first schema is in BCNF, the second one may not be (and may require further decomposition)
- No repetition now: \( r_1 \) contains \((a_1, b_1)\), but \( b_1 \) will not be repeated

Careful: you want the decomposition to be **lossless**
- No information should be lost
  - The above decomposition is lossless

---

### Achieving BCNF Schemas

- For all dependencies \( \alpha \rightarrow \beta \) in \( F^+ \), check if \( A \) is a superkey
  - By using attribute closure

- If not, then
  - Choose a dependency in \( F^+ \) that breaks the BCNF rules, say \( \alpha \rightarrow \beta \)
  - Create \( R_1 = \alpha \beta \)
  - Create \( R_2 = \alpha (R - \beta - \alpha) = R - \beta \)
  - Note that: \( R_1 \cap R_2 = \alpha \) and \( \alpha \rightarrow \alpha \beta \) (= \( R_1 \)), so this is lossless decomposition

- Repeat for \( R_1 \), and \( R_2 \)
  - By defining \( F_1 \) to be all dependencies in \( F \) that contain only attributes in \( R_1 \)
  - Similarly \( F_2 \)
Example 1

\[ R = (A, B, C) \]
\[ F = \{A \rightarrow B, B \rightarrow C\} \]
Candidate keys = \{A\}
BCNF? No. B \rightarrow C violates.

\[ B \rightarrow C \]

\[ R1 = (B, C) \]
\[ F1 = \{B \rightarrow C\} \]
Candidate keys = \{B\}
BCNF = true

\[ R2 = (A, B) \]
\[ F2 = \{A \rightarrow B\} \]
Candidate keys = \{A\}
BCNF = true

\[ R3 = (A, B) \]
\[ F3 = \{A \rightarrow B\} \]
Candidate keys = \{A\}
BCNF = true

\[ BC \rightarrow D \]

\[ R1 = (B, C, D) \]
\[ F1 = \{BC \rightarrow D\} \]
Candidate keys = \{BC\}
BCNF = true

\[ R2 = (B, C, A, E) \]
\[ F2 = \{A \rightarrow B\} \]
Candidate keys = \{ACE\}
BCNF = false (A \rightarrow B)

\[ A \rightarrow B \]

\[ R3 = (A, B) \]
\[ F3 = \{A \rightarrow B\} \]
Candidate keys = \{A\}
BCNF = true

\[ R4 = (A, C, E) \]
\[ F4 = \{\} \] [[ only trivial ]] 
Candidate keys = \{ACE\}
BCNF = true

Dependency preservation ???
We can check:
BC \rightarrow D (R1), A \rightarrow B (R3),
Dependency-preserving decomposition
**R** = (A, B, C, D, E) 
**F** = {A → B, BC → D}  
Candidate keys = {ACE}  
BCNF = Violated by {A → B, BC → D} etc…

From A → B  
AC → BC (aug)  
AC → D (trans)

**R1** = (A, B)  
**F1** = {A → B}  
Candidate keys = {A}  
BCNF = true

**R2** = (A, C, D, E)  
**F2** = {AC → D}  
Candidate keys = {ACE}  
BCNF = false (AC → D)

**R3** = (A, C, D)  
**F3** = {AC → D}  
Candidate keys = {AC}  
BCNF = true

**R4** = (A, C, E)  
**F4** = {} [[ only trivial ]]  
Candidate keys = {ACE}  
BCNF = true

**Dependency preservation ???**  
We can check:  
A → B (R1), AC → D (R3),  
but we lost BC → D  
So this is not a dependency-preserving decomposition

**R** = (A, B, C, D, E, H)  
**F** = {A → BC, E → HA}  
Candidate keys = {DE}  
BCNF = Violated by {A → BC} etc…

**R1** = (A, B, C)  
**F1** = {A → BC}  
Candidate keys = {A}  
BCNF = true

**R2** = (A, D, E, H)  
**F2** = {E → HA}  
Candidate keys = {DE}  
BCNF = false (E → HA)

**R3** = (E, H, A)  
**F3** = {E → HA}  
Candidate keys = {E}  
BCNF = true

**R4** = (ED)  
**F4** = {} [[ only trivial ]]  
Candidate keys = {DE}  
BCNF = true

Dependency preservation ???  
We can check:  
A → BC (R1), E → HA (R3),  
Dependency-preserving decomposition
2. Closure of an attribute set

- Given a set of attributes $\alpha$ and a set of FDs $F$,
- **closure of $\alpha$ under $F$** is the set of all attributes implied by $\alpha$

- In other words, the largest $\beta$ such that: $\alpha \rightarrow \beta$

Redefining **super keys**:
- The closure of a super key is the entire relation schema

Redefining **candidate keys**:
- 1. It is a super key
- 2. No subset of it is a super key

Computing the closure for $\alpha$

- Simple algorithm

  1. Start with $\beta = \alpha$.
  2. Go over all functional dependencies, $\delta \rightarrow \gamma$, in $F^+$
  3. If $\delta \subseteq \beta$, then
      Add $\gamma$ to $\beta$
  4. Repeat till $\beta$ stops changing
Example

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

- \((AG)^+\)?
  - 1. result = AG
  - 2. result = ABCG \( (A \rightarrow C \text{ and } A \rightarrow B) \)
  - 3. result = ABCGH \( (CG \rightarrow H \text{ and } CG \subseteq AGBC) \)
  - 4. result = ABCGHI \( (CG \rightarrow I \text{ and } CG \subseteq AGBCH) \)

- Is \((AG)\) a candidate key?
  - 1. It is a super key.
  - 2. \((A^+) = ABCH, (G^+) = G\).
    - YES.