CMSC424: Normalization

Instructor: Amol Deshpande
amol@cs.umd.edu

Databases

- Data Models
  - Conceptual representation of the data
- Data Retrieval
  - How to ask questions of the database
  - How to answer those questions
- Data Storage
  - How/where to store data, how to access it
- Data Integrity
  - Manage crashes, concurrency
  - Manage semantic inconsistencies
Relational Database Design

- Where did we come up with the schema that we used?
  - E.g. why not store the actor names with movies?

- If from an E-R diagram, then:
  - Did we make the right decisions with the E-R diagram?

- Goals:
  - Formal definition of what it means to be a “good” schema.
  - How to achieve it.

Movies Database Schema

Movie(title, year, length, inColor, studioName, producerC#)
StarsIn(movieTitle, movieYear, starName)
MovieStar(name, address, gender, birthdate)
MovieExec(name, address, cert#, netWorth)
Studio(name, address, presC#)

Changed to:

Movie(title, year, length, inColor, studioName, producerC#, starName)
<StarsIn merged into above>
MovieStar(name, address, gender, birthdate)
MovieExec(name, address, cert#, netWorth)
Studio(name, address, presC#)

Is this a good schema ???
## Movie Database

<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>2005</td>
<td>187</td>
<td>Yes</td>
<td>Universal</td>
<td>150</td>
<td>Watts</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>

### Issues:

1. **Redundancy** → higher storage, inconsistencies (“anomalies”)
   - update anomalies, insertion anomalies
2. **Need nulls**
   - Unable to represent some information without using nulls
   
   _How to store movies w/o actors (pre-productions etc) ?_

### Movie Database

<table>
<thead>
<tr>
<th>Title</th>
<th>Year</th>
<th>Length</th>
<th>inColor</th>
<th>StudioName</th>
<th>prodC#</th>
<th>starNames</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, Fisher, H. Ford}</td>
</tr>
<tr>
<td>King Kong</td>
<td>2005</td>
<td>187</td>
<td>Yes</td>
<td>Universal</td>
<td>150</td>
<td>Watts</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>

### Issues:

3. **Avoid sets**
   - Hard to represent
   - Hard to query
Smaller schemas always good ????

Split Studio\( (\text{name}, \text{address}, \text{presC#}) \) into:

<table>
<thead>
<tr>
<th>Studio1 (\text{name}, \text{presC#})</th>
<th>Studio2(\text{name}, \text{address})</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>\text{Name}</td>
<td>\text{Name}</td>
</tr>
<tr>
<td>Fox</td>
<td>Fox</td>
</tr>
<tr>
<td>Studio2</td>
<td>Studio2</td>
</tr>
<tr>
<td>Universal</td>
<td>Universal</td>
</tr>
</tbody>
</table>

|                                      | \text{Address}                       |
|                                      |                                        |
|                                      | Fox                                   |
|                                      | Address1                             |
|                                      |                                        |
|                                      | Studio2                               |
|                                      | Address1                             |
|                                      |                                        |
|                                      | Universal                            |
|                                      | Address2                             |

This process is also called “decomposition”

**Issues:**

4. Requires more joins (w/o any obvious benefits)
5. Hard to check for some dependencies

What if the “address” is actually the presC#'s address?

No easy way to ensure that constraint (w/o a join).

---

Smaller schemas always good ????

Decompose StarsIn\( (\text{movieTitle}, \text{movieYear}, \text{starName}) \) into:

<table>
<thead>
<tr>
<th>StarsIn1(\text{movieTitle}, \text{movieYear})</th>
<th>StarsIn2(\text{movieTitle}, \text{starName})</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>\text{movieTitle}</td>
<td>\text{movieTitle}</td>
</tr>
<tr>
<td>Star wars</td>
<td>Star Wars</td>
</tr>
<tr>
<td>King Kong</td>
<td>King Kong</td>
</tr>
<tr>
<td>King Kong</td>
<td>King Kong</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>\text{movieYear}</td>
<td>\text{starName}</td>
</tr>
<tr>
<td>1977</td>
<td>Hamill</td>
</tr>
<tr>
<td>1933</td>
<td>Watts</td>
</tr>
<tr>
<td>2005</td>
<td>Faye</td>
</tr>
</tbody>
</table>

**Issues:**

6. “joining” them back results in more tuples than what we started with

\( \text{(King Kong, 1933, Watts)} \) & \( \text{(King Kong, 2005, Faye)} \)

This is a “lossy” decomposition

We lost some constraints/information

The previous example was a “lossless” decomposition.
Desired data

- No sets
- Correct and faithful to the original design
  - Avoid lossy decompositions
- As little redundancy as possible
  - To avoid potential anomalies
- No “inability to represent information”
  - Nulls shouldn’t be required to store information
- Dependency preservation
  - Should be possible to check for constraints

Not always possible. We sometimes relax these for: simpler schemas, and fewer joins during queries.

Approach

1. We will encode and list all our knowledge about the schema
   - Functional dependencies (FDs)
     SSN \(\rightarrow\) name  (means: SSN “implies” length)
     - If two tuples have the same “SSN”, they must have the same “name”
       movietitle \(\rightarrow\) length ??? Not true.
     - But, (movietitle, movieYear) \(\rightarrow\) length --- True.
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
Functional Dependencies

- Let $R$ be a relation schema and 
  $\alpha \subseteq R$ and $\beta \subseteq R$
- 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 $\beta$.
  $t_1[\alpha] = t_2[\alpha] \Rightarrow t_1[\beta] = t_2[\beta]$
- Example:

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>4</td>
</tr>
<tr>
<td>1</td>
<td>5</td>
</tr>
<tr>
<td>3</td>
<td>7</td>
</tr>
</tbody>
</table>

- On 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 relation

<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 to use domain knowledge to decide which FDs hold
Functional Dependencies

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

- Deriving FDs
  - A set of FDs may imply other FDs
  - e.g. If $A \rightarrow B$, and $B \rightarrow C$, then clearly $A \rightarrow C$
  - 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, $A \rightarrow B$, is called trivial if:
   - B is a subset of A
   - e.g. Movieyear, length $\rightarrow$ 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 $A \rightarrow B$ that holds on it is EITHER:
    1. Trivial OR
    2. $A$ 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
Why does redundancy arise?

Given a FD, $A \rightarrow B$, if $A$ is repeated $(B - A)$ has to be repeated
1. If rule 1 is satisfied, $(B - A)$ is empty, so not a problem.
2. If rule 2 is satisfied, then $A$ 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
BCNF

- 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 $(a1, b1)$, but $b1$ will not be repeated

- Careful: you want the decomposition to be lossless
  - No information should be lost
    - The above decomposition is lossless
  - We will define this more formally later

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
1. Closure

- Given a set of functional dependencies, $F$, its closure, $F^+$, is all FDs that are implied by FDs in $F$.
  - e.g. If $A \rightarrow B$, and $B \rightarrow C$, then clearly $A \rightarrow C$

- We can find $F^+$ by applying Armstrong’s Axioms:
  - if $\beta \subseteq \alpha$, then $\alpha \rightarrow \beta$ (reflexivity)
  - if $\alpha \rightarrow \beta$, then $\gamma \alpha \rightarrow \gamma \beta$ (augmentation)
  - if $\alpha \rightarrow \beta$, and $\beta \rightarrow \gamma$, then $\alpha \rightarrow \gamma$ (transitivity)

- These rules are
  - sound (generate only functional dependencies that actually hold)
  - complete (generate all functional dependencies that hold)

Additional rules

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

- The above rules can be inferred from Armstrong’s axioms.
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 augmenting \( CG \rightarrow I \) to infer \( CG \rightarrow CGI \),
      and augmenting of \( CG \rightarrow H \) to infer \( CGI \rightarrow HI \),
      and then transitivity

2. Closure of an attribute set

- Given a set of attributes \( A \) and a set of FDs \( F \), closure of \( A \) under \( F \) is the set of all attributes implied by \( A \)
- In other words, the largest \( B \) such that: \( A \rightarrow B \)
- 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 $A$

- Simple algorithm
  1. Start with $B = A$.
  2. Go over all functional dependencies, $\beta \rightarrow \gamma$, in $F^+$
  3. If $\beta \subseteq B$, then
     Add $\gamma$ to $B$
  4. Repeat till $B$ changes

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. $(A^+) = BC$, $(G^+) = G.$

YES.
Uses of attribute set closures

- Determining superkeys and candidate keys

- Determining if $A \rightarrow B$ is a valid FD
  - Check if $A^+$ contains $B$

- Can be used to compute $F^+$

3. Extraneous Attributes

- Consider $F$, and a functional dependency, $A \rightarrow B$.

- “Extraneous”: Are there any attributes in $A$ or $B$ that can be safely removed?  
  *Without changing the constraints implied by $F*  

- Example: Given $F = \{A \rightarrow C, AB \rightarrow CD\}$
  - $C$ is extraneous in $AB \rightarrow CD$ since $AB \rightarrow C$ can be inferred even after deleting $C$
  - i.e., given: $A \rightarrow C$, and $AB \rightarrow D$, we can use Armstrong Axioms to infer $AB \rightarrow CD$
4. Canonical Cover

- A **canonical cover** for $F$ is a set of dependencies $F_c$ such that
  - $F$ logically implies all dependencies in $F_c$, and
  - $F_c$ logically implies all dependencies in $F$, and
  - No functional dependency in $F_c$ contains an extraneous attribute, and
  - Each left side of functional dependency in $F_c$ is unique

- In some (vague) sense, it is a *minimal* version of $F$

- Read up algorithms to compute $F_c$

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
Loss-less Decompositions

- Definition: A decomposition of $R$ into $(R_1, R_2)$ is called lossless if, for all legal instance of $r(R)$:
  \[ r = \prod_{R_1} (r) \bowtie \prod_{R_2} (r) \]

- In other words, projecting on $R_1$ and $R_2$, and joining back, results in the relation you started with.

- Rule: A decomposition of $R$ into $(R_1, R_2)$ is lossless, iff:
  \[ R_1 \cap R_2 \rightarrow R_1 \quad \text{or} \quad R_1 \cap R_2 \rightarrow R_2 \]
in $F^+$.

Dependency-preserving Decompositions

Is it easy to check if the dependencies in $F$ hold?

Okay as long as the dependencies can be checked in the same table.

Consider $R = (A, B, C)$, and $F = \{A \rightarrow B, B \rightarrow C\}$

1. Decompose into $R_1 = (A, B)$, and $R_2 = (A, C)$
   - Lossless? Yes.
   - But, makes it hard to check for $B \rightarrow C$
     - The data is in multiple tables.

2. On the other hand, $R_1 = (A, B)$, and $R_2 = (B, C)$,
   - is both lossless and dependency-preserving
   - Really? What about $A \rightarrow C$?
   - If we can check $A \rightarrow B$, and $B \rightarrow C$, $A \rightarrow C$ is implied.
Definition:
- Consider decomposition of $R$ into $R_1, ..., R_n$.
- Let $F_i$ be the set of dependencies $F^+$ that include only attributes in $R_i$.

The decomposition is **dependency preserving**, if
$$(F_1 \cup F_2 \cup ... \cup F_n)^+ = F^+$$

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
**BCNF**

- Given a relation schema $R$, and a set of functional dependencies $F$, if every FD, $A \rightarrow B$, is either:
  1. Trivial
  2. $A$ is a superkey of $R$

  Then, $R$ is in **BCNF (Boyce-Codd Normal Form)**

- What if the schema is not in BCNF?
  - Decompose (split) the schema into two pieces.
  - Careful: you want the decomposition to be lossless

---

**Achieving BCNF Schemas**

For all dependencies $A \rightarrow B$ 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 $A \rightarrow B$
   - Create $R1 = A \ B$
   - Create $R2 = A (R - B - A)$
   - Note that: $R1 \cap R2 = A$ and $A \rightarrow AB (= R1)$, so this is lossless decomposition

Repeat for $R1$, and $R2$

By defining $F1+$ to be all dependencies in $F$ that contain only attributes in $R1$

Similarly $F2+$
Example 1

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

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

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

Example 2-1

\[ R = (A, B, C, D, E) \]
\[ F = \{A \rightarrow B, BC \rightarrow D\} \]
Candidate keys = \{ACE\}
BCNF = Violated by \{A \rightarrow B, BC \rightarrow D\} etc…

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

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

From A \rightarrow B and BC \rightarrow D by pseudo-transitivity

Dependency preservation ???
We can check:
\[ A \rightarrow B \text{ (R1), } AC \rightarrow D \text{ (R3),} \]

but we lost BC \rightarrow D
So this is not a dependency-preserving decomposition
Example 2-2

\[ R = (A, B, C, D, E) \]
\[ F = \{ A \rightarrow B, BC \rightarrow D \} \]
Candidate keys = \{ACE\}
BCNF = Violated by \{A \rightarrow B, BC \rightarrow D\} etc…

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

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

Dependency preservation ???
We can check:
BC \rightarrow D (R1), A \rightarrow B (R3),
Dependency-preserving decomposition

Example 3

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

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

\[ R2 = (A, D, E, H) \]
\[ F2 = \{ E \rightarrow HA \} \]
Candidate keys = \{DE\}
BCNF = false (E \rightarrow HA)

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

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

Dependency preservation ???
We can check:
A \rightarrow BC (R1), E \rightarrow HA (R3),
Dependency-preserving decomposition
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

Outline

BCNF may not preserve dependencies

$R = \{ J, K, L \}$

$F = \{ JK \rightarrow L, L \rightarrow K \}$

Two candidate keys = $JK$ and $JL$

$R$ is not in BCNF

Any decomposition of $R$ will fail to preserve

$JK \rightarrow L$

This implies that testing for $JK \rightarrow L$ requires a join
BCNF may not preserve dependencies

- Not always possible to find a dependency-preserving decomposition that is in BCNF.
- PTIME to determine if there exists a dependency-preserving decomposition in BCNF
  - in size of $F$
- NP-Hard to find one if it exists
- Better results exist if $F$ satisfies certain properties

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
3NF

- **Definition: Prime attributes**
  An attribute that is contained in a candidate key for R

- **Example 1:**
  - $R = \{A, B, C, D, E, H\}, F = \{A \rightarrow BC, E \rightarrow HA\}$
  - Candidate keys = \{ED\}
  - Prime attributes: D, E

- **Example 2:**
  - $R = \{J, K, L\}, F = \{JK \rightarrow L, L \rightarrow K\}$
  - Candidate keys = \{JL, JK\}
  - Prime attributes: J, K, L

- **Observation/Intuition:**
  1. A key has no redundancy (is not repeated in a relation)
  2. A prime attribute has limited redundancy

---

3NF

- Given a relation schema $R$, and a set of functional dependencies $F$, if every FD, $A \rightarrow B$, is either:
  1. Trivial, or
  2. $A$ is a superkey of $R$, or
  3. All attributes in $(B - A)$ are prime

Then, $R$ is in **3NF (3rd Normal Form)**

- *Why is 3NF good?*
3NF and Redundancy

Why does redundancy arise?
- Given a FD, A → B, if A is repeated (B – A) has to be repeated
  1. If rule 1 is satisfied, (B – A) is empty, so not a problem.
  2. If rule 2 is satisfied, then A can’t be repeated, so this doesn’t happen either.
  3. If not, rule 3 says (B – A) must contain only prime attributes.
    This limits the redundancy somewhat.

- So 3NF relaxes BCNF somewhat by allowing for some (hopefully limited) redundancy.
- Why?
  - There always exists a dependency-preserving lossless decomposition in 3NF.

Decomposing into 3NF

- A synthesis algorithm
- Start with the canonical cover, and construct the 3NF schema directly
- Homework assignment.
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

BCNF and redundancy

<table>
<thead>
<tr>
<th>MovieTitle</th>
<th>MovieYear</th>
<th>StarName</th>
<th>Address</th>
</tr>
</thead>
<tbody>
<tr>
<td>Star wars</td>
<td>1977</td>
<td>Harrison Ford</td>
<td>Address 1, LA</td>
</tr>
<tr>
<td>Star wars</td>
<td>1977</td>
<td>Harrison Ford</td>
<td>Address 2, FL</td>
</tr>
<tr>
<td>Indiana Jones</td>
<td>198x</td>
<td>Harrison Ford</td>
<td>Address 1, LA</td>
</tr>
<tr>
<td>Indiana Jones</td>
<td>198x</td>
<td>Harrison Ford</td>
<td>Address 2, FL</td>
</tr>
<tr>
<td>Witness</td>
<td>19xx</td>
<td>Harrison Ford</td>
<td>Address 1, LA</td>
</tr>
<tr>
<td>Witness</td>
<td>19xx</td>
<td>Harrison Ford</td>
<td>Address 2, FL</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

Lot of redundancy

FDs? No non-trivial FDs.

So the schema is trivially in BCNF (and 3NF)

What went wrong?
Multi-valued Dependencies

- The redundancy is because of multi-valued dependencies
- Denoted:
  - `starnname →→ address`
  - `starnname →→ movietitle, moviyear`

- Should not happen if the schema is constructed from an E/R diagram

- Functional dependencies are a special case of multi-valued dependencies

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
4NF

- Similar to BCNF, except with MVDs instead of FDs.

- Given a relation schema $R$, and a set of multi-valued dependencies $F$, if every MVD, $A \rightarrow\rightarrow B$, is either:
  1. Trivial, or
  2. $A$ is a superkey of $R$

Then, $R$ is in \textit{4NF (4th Normal Form)}

- $4NF \rightarrow BCNF \rightarrow 3NF \rightarrow 2NF \rightarrow 1NF$:
  - If a schema is in 4NF, it is in BCNF.
  - If a schema is in BCNF, it is in 3NF.

- Other way round is untrue.

### Comparing the normal forms

<table>
<thead>
<tr>
<th></th>
<th>3NF</th>
<th>BCNF</th>
<th>4NF</th>
</tr>
</thead>
<tbody>
<tr>
<td>Eliminates redundancy because of FD's</td>
<td>Mostly</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Eliminates redundancy because of MVD's</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>Preserves FDs</td>
<td>Yes</td>
<td>Maybe</td>
<td>Maybe</td>
</tr>
<tr>
<td>Preserves MVDs</td>
<td>Maybe</td>
<td>Maybe</td>
<td>Maybe</td>
</tr>
</tbody>
</table>

4NF is typically desired and achieved.

A good E/R diagram won't generate non-4NF relations at all

Choice between 3NF and BCNF is up to the designer
Database design process

- Three ways to come up with a schema
  1. Using E/R diagram
     - If good, then little normalization is needed
     - Tends to generate 4NF designs
  2. A universal relation $R$ that contains all attributes.
     - Called universal relation approach
     - Note that MVDs will be needed in this case
  3. An *ad hoc* schema that is then normalized
     - MVDs may be needed in this case

Recap

- What about 1st and 2nd normal forms?

  1NF:
  - Essentially says that no set-valued attributes allowed
  - Formally, a domain is called *atomic* if the elements of the domain are considered indivisible
  - A schema is in 1NF if the domains of all attributes are atomic
  - We assumed 1NF throughout the discussion
    - Non 1NF is just not a good idea

  2NF:
  - Mainly historic interest
  - See Exercise 7.15 in the book
Recap

- We would like our relation schemas to:
  - Not allow potential redundancy because of FDs or MVDs
  - Be *dependency-preserving*:
    - Make it easy to check for dependencies
    - Since they are a form of integrity constraints
- Functional Dependencies/Multi-valued Dependencies
  - Domain knowledge about the data properties
- Normal forms
  - Defines the rules that schemas must follow
  - 4NF is preferred, but 3NF is sometimes used instead

Recap

- Denormalization
  - After doing the normalization, we may have too many tables
  - We may *denormalize* for performance reasons
    - Too many tables $\Rightarrow$ too many joins during queries
  - A better option is to use *views* instead
    - So if a specific set of tables is joined often, create a view on the join
- More advanced normal forms
  - project-join normal form (PJNF or 5NF)
  - domain-key normal form
  - Rarely used in practice