Normalization

BCNF

- Given a relation schema $R$, and a set of functional dependencies $F$, if every FD, $\alpha \rightarrow \beta$, is either:
  1. Trivial
  2. $\alpha$ 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 $\alpha \rightarrow \beta$ 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 $R_2 = \alpha(R - \beta - \alpha)$
  - Note that: $R_1 \cap R_2 = \alpha$ and $\alpha \rightarrow \alpha \beta$ ($= R_1$), so this is lossless decomposition

- Repeat for $R_1, R_2$
  - 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.

$B \rightarrow C$

$R_1 = (B, C)$
$F_1 = \{B \rightarrow C\}$
Candidate keys = $\{B\}$
BCNF = true

$R_2 = (A, B)$
$F_2 = \{A \rightarrow B\}$
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)

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

\[ BC \rightarrow D \]

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

\[ A \rightarrow B \]

\[ R2 = (B, C, A, E) \]
\[ F2 = \{A \rightarrow B\} \]
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 ]]\nCandidate keys = \{DE\}
BCNF = true

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

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 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 → BC, E → HA},
  - Candidate keys = {ED}
  - Prime attributes: D, E

- Example 2:
  - R = (J, K, L), F = {JK → L, L → 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**

\[ R \text{ is in } 3NF \ (3^{rd} \text{ Normal Form}) \text{ if:} \]

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

**Why is 3NF good?**
- Preserves dependencies.

---

**3NF 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
    3. If not, rule 3 says \( (\beta - \alpha) \) must contain only *prime attributes*
      - This limits the redundancy somewhat.

- 3NF relaxes BCNF by allowing some (hopefully limited) redundancy
- Why good?
  - There always exists a dependency-preserving lossless decomposition in 3NF.
**Decomposing into 3NF**

let $F_\circ$ be a canonical cover for $F$

$i := 0$

for each functional dependency $\alpha \rightarrow \beta$ in $F_\circ$

$i := i + 1$

$R_i := \alpha \beta$

if none of the schemas $R_j, j = 1, 2, \ldots, i$ contains a candidate key for $R$

then

$i := i + 1$

$R_i := any\ candidate\ key\ for\ R$

else

/* Optionally, remove redundant relations */

repeat

if any schema $R_j$ is contained in another schema $R_k$

then

/* Delete $R_j$ */

$R_j := R_k$

$i := i - 1$

until no more $R_j$s can be deleted

return $(R_1, R_2, \ldots, R_i)$

**Figure 8.12** Dependency-preserving, lossless decomposition into 3NF.

---

**3CNF Example**

- $(R) = \{A,B,C,D,E,F,G,H\}$

- Function Dependencies
  - $F = \{A \rightarrow CGH, AD \rightarrow C, DE \rightarrow F, G \rightarrow H\}$
    - $R_1 = \{ACGH\}$, $R_2 = \{ADC\}$, $R_3 = \{DEF\}$, $R_4 = \{GH\}$
    - $R_1 = \{ACGH\}$, $R_2 = \{ADC\}$, $R_3 = \{DEF\}$, $R_4 = \{GH\}$
    - $R_5 = \{ABDE\}$
    - $R_1 = \{ACGH\}$, $R_2 = \{ADC\}$, $R_3 = \{DEF\}$, $R_4 = \{GH\}$
    - $R_5 = \{ABDE\}$
  - $F' =$
    - $\{A \rightarrow CGH, AD \rightarrow C, DE \rightarrow F, G \rightarrow H\}$
    - $\{A \rightarrow CGH, AD \rightarrow C, DE \rightarrow F, G \rightarrow H\}$
    - $\{A \rightarrow CG, DE \rightarrow F, G \rightarrow H\}$
      - H is extra in A \rightarrow CGH, D extra in AD \rightarrow C - then merge w/ A \rightarrow CG
      - $R_1 = \{ACG\}$, $R_2 = \{DEF\}$, $R_3 = \{GH\}$
    - $R_1 = \{ACG\}$, $R_2 = \{DEF\}$, $R_3 = \{GH\}$
    - $R_4 = \{ABDE\}$
  - Lossless: Each has a single FD that is a key
  - Preserves dependencies: each carried through a single subrelation
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 (or maybe one) 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:**
  
  \[ \text{starname} \rightarrow \rightarrow \text{address} \]
  
  \[ \text{starname} \rightarrow \rightarrow \text{movietitle, moviyear} \]

- 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, \( A \rightarrow B \), is either:
  1. Trivial, or
  2. \( A \) is a superkey of \( R \)

Then, \( R \) is in \( 4NF \) (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 1$^{st}$ and 2$^{nd}$ 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
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

Denormalization
- After doing the normalization, we may have too many tables
- We may denormalize for performance reasons
  - Too many tables → 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