3. Extraneous Attributes

- Example: Given \( F = \{ A \rightarrow C, AB \rightarrow CD \} \), show \( C \) extra in \( AB \rightarrow CD \)
  - \( F' = \{ A \rightarrow C, AB \rightarrow D \} \)

  - Using Armstrong’s :
    (show \( F' \rightarrow F \))
    - We know:
      - \( AB \rightarrow D \) \( (F') \)
      - \( ABC \rightarrow CD \) \( (\text{aug}) \)
    - also:
      - \( A \rightarrow C \) \( (F') \)
      - \( AB \rightarrow BC \) \( (\text{aug w/ B}) \)
      - \( AB \rightarrow ABC \) \( (\text{aug w/ A}) \)
    - then:
      - \( AB \rightarrow ABC \rightarrow CD \) \( \text{(trans)} \)
      - done.

\[ \sigma \text{ is extraneous in } \alpha \text{ iff:} \]
\[ F \rightarrow F', \text{ or} \]
\[ (\alpha - \sigma)^+ \text{ includes } \beta \text{ under } F \]

\[ \sigma \text{ is extraneous in } \beta \text{ iff:} \]
\[ F' \rightarrow F, \text{ or} \]
\[ \alpha^+ \text{ includes } \sigma \text{ in } F' \]
3. Extraneous Attributes

- Example: Given $F = \{A \rightarrow C, AB \rightarrow CD\}$, show $C$ extra in $AB \rightarrow CD$
  - $F' = \{A \rightarrow C, AB \rightarrow D\}$

  - Using Armstrong’s:
    (show $F' \rightarrow F$)
    - We know:
      - $AB \rightarrow D$ (F’)
      - $ABC \rightarrow CD$ (aug)
    - also:
      - $A \rightarrow C$ (F’)
      - $AB \rightarrow BC$ (aug w/ $B$)
      - $AB \rightarrow ABC$ (aug w/ $A$)
    - then:
      - $AB \rightarrow ABC \rightarrow CD$ (trans)
    done.

  - Attribute closures (show $\alpha^+ \text{ includes } C$ under $F'$):
    - $(AB)^+ = AB$
    - $= ABC \quad (A \rightarrow C)$
    done.

\[
\sigma \text{ is extraneous in } \alpha \text{ iff:} \\
F \rightarrow F', \text{ or} \\
(\alpha - \sigma)^+ \text{ includes } \beta \text{ under } F
\]

\[
\sigma \text{ is extraneous in } \beta \text{ iff:} \\
F' \rightarrow F, \text{ or} \\
\alpha^+ \text{ includes } \sigma \text{ in } F'
\]
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 $F$

- Create as follows:
  - **repeat**
    1. use union rule to merge right sides
    2. eliminate extraneous attributes
  - **until $F_c$ does not change**

---

### Example

- A → B, A → C, C → D, AC → BD

- **Cover:**
  1. A → BC, C → D, AC → BD
  2. A → BCD, C → D

- **σ is extraneous in α iff:**
  - $F \rightarrow F'$, or
  - $(\alpha - \sigma)^+ \text{ includes } \beta \text{ under } F$

- **σ is extraneous in β iff:**
  - $F' \rightarrow F$, or
  - $\alpha^+ \text{ includes } \sigma \text{ in } F'$

- $F_c = A \rightarrow BC, C \rightarrow D$
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 lossless if, for all legal instance of $r(R)$:
  
  $$
  r = \prod_{R_1} (r) \bowtie \prod_{R_2} (r) ((\text{select all } R_1) \text{ join } (\text{select all } R_2))
  $$
  
  or
  
  $$(\text{select } * \text{ from (select } R_1 \text{ from } r \text{) natural join (select } R_2 \text{ from } r))$$

- In other words, projecting on $R_1$ and $R_2$, and joining back, gives the original relation

- Rule: A decomposition of $R$ into $(R_1, R_2)$ is lossless, iff:
  
  $$
  R_1 \cap R_2 \rightarrow R_1 \text{ or } R_1 \cap R_2 \rightarrow R_2
  $$

  in $F^+$. 

  $(R_1 \cap R_2)$ must be key for $R_1$ or $R_2$
Dependency-preserving Decompositions

- Is it easy to check if dependencies in $F$ hold?
  - Yes if dependencies can be checked in the same table.
- Consider $R = (A, B, C)$, and $F = \{ A \rightarrow B, B \rightarrow C \}$
  - Decompose into $R_1 = (A, B)$, and $R_2 = (A, C)$
    - Lossless?
      - Yes: $AB \cap AC = A$, which is a key for $R_1$
      - But harder to check for $B \rightarrow C$ as the data is in multiple tables.
  - Decompose into $R_1 = (A, B)$, and $R_2 = (B, C)$,
    - is both lossless and dependency-preserving

Definition:
- Consider decomposition of $R$ into $R_1, \ldots, R_n$.
- Let $F_i$ be dependencies using just attributes in $R_i$.

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

Normalization
Recall that $R$ is in BCNF if every FD, $\alpha \rightarrow \beta$, is either:

1. Trivial, or
2. $\alpha$ is a superkey of $R$

No redundancy

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 $\alpha$ is a superkey

- (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 $R_2 = R - (\beta - \alpha)$.
- Note that: $R_1 \cap R_2 = \alpha$ and $\alpha \rightarrow \alpha\beta$, so:
  - $\alpha$ is a superkey of $R_1$
  - lossless decomposition (lossless if intersection of two attribute sets is key for one)

Repeat for $R_1$ and $R_2$

- Define $F_i$ to be all dependencies in $F^+$ that contain only attributes in $R_i$
Achieving BCNF Schemas

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

• Dependency-preserving?
  • yes

Example 2a

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

\[ A \rightarrow B \]

\[ R_1 = (A, B) \]
\[ F_1 = \{A \rightarrow B\} \]
Candidate keys = \{A\}
BCNF = true

\[ R_2 = (A, C, D, E) \]
\[ F_2 = \{} \]
Candidate keys = \{ACDE\}
BCNF = true

• Dependency-preserving?
  • no: lost \( BC \rightarrow D \)
Example 2b

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

\[ BC \rightarrow D \]

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

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

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\} and \{E \rightarrow HA\}

\[ A \rightarrow BC \]

\[ R1 = (A, B, C) \]
\[ F1 = \{ A \rightarrow BC \} \]
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-preserving?
  • yes