Functional Dependencies

Prof. Chris Clifton
28 September 2016

Relational Design

Instructor
ID
Name
Salary

Dept
dept_name
building
budget

Works_in

Instructor(ID number(10) primary key,
Name varchar(40),
Salary number(6))

Dept(dept_name varchar(20) primary key,
building varchar(30),
budget number(8))

Works_in(ID references Instructor(ID),
department references Dept(dept_name))

Key for Works_in?
A. ID
B. dept_name
C. both
D. neither
Keys of Relations

\( K \) is a key for relation \( R \) if:
1. \( K \rightarrow \text{all attributes of } R \). (Uniqueness)
2. For no proper subset of \( K \) is (1) true. (Minimality)
   - If \( K \) at least satisfies (1), then \( K \) is a superkey.

Conventions
- Pick one key; underline key attributes in the relation schema.
- \( X \), etc., represent sets of attributes; \( A \) etc., represent single attributes.

Combine Schemas?

- Suppose we combine instructor and department into \( \text{inst_dept} \)

<table>
<thead>
<tr>
<th>ID</th>
<th>name</th>
<th>salary</th>
<th>dept_name</th>
<th>building</th>
<th>budget</th>
</tr>
</thead>
<tbody>
<tr>
<td>22222</td>
<td>Einstein</td>
<td>95000</td>
<td>Physics</td>
<td>Watson</td>
<td>70000</td>
</tr>
<tr>
<td>12121</td>
<td>Wu</td>
<td>90000</td>
<td>Finance</td>
<td>Painter</td>
<td>120000</td>
</tr>
<tr>
<td>32343</td>
<td>El Said</td>
<td>60000</td>
<td>History</td>
<td>Painter</td>
<td>50000</td>
</tr>
<tr>
<td>45565</td>
<td>Katz</td>
<td>75000</td>
<td>Comp. Sci.</td>
<td>Taylor</td>
<td>100000</td>
</tr>
<tr>
<td>98345</td>
<td>Kim</td>
<td>80000</td>
<td>Elec. Eng.</td>
<td>Taylor</td>
<td>85000</td>
</tr>
<tr>
<td>76766</td>
<td>Crick</td>
<td>72000</td>
<td>Biology</td>
<td>Watson</td>
<td>90000</td>
</tr>
<tr>
<td>10101</td>
<td>Srinivasan</td>
<td>65000</td>
<td>Comp. Sci.</td>
<td>Taylor</td>
<td>100000</td>
</tr>
<tr>
<td>58583</td>
<td>Califieri</td>
<td>62000</td>
<td>History</td>
<td>Painter</td>
<td>50000</td>
</tr>
<tr>
<td>83821</td>
<td>Brandt</td>
<td>92000</td>
<td>Comp. Sci.</td>
<td>Taylor</td>
<td>100000</td>
</tr>
<tr>
<td>15151</td>
<td>Mozart</td>
<td>40000</td>
<td>Music</td>
<td>Packard</td>
<td>80000</td>
</tr>
<tr>
<td>33456</td>
<td>Gold</td>
<td>87000</td>
<td>Physics</td>
<td>Watson</td>
<td>70000</td>
</tr>
<tr>
<td>76543</td>
<td>Singh</td>
<td>80000</td>
<td>Finance</td>
<td>Painter</td>
<td>120000</td>
</tr>
</tbody>
</table>

- This will
  A. Duplicate columns
  B. Duplicate data
  C. Save space
A Combined Schema Without Repetition

- Consider combining relations
  - `sec_class(sec_id, building, room_number)` and
  - `section(course_id, sec_id, semester, year)`
  into one relation
    - `section(course_id, sec_id, semester, year, building, room_number)`
- No repetition in this case

What About Smaller Schemas?

- Suppose we had started with `inst_dept`. How would we know to split up (decompose) it into `instructor` and `department`?
- Write a rule “if there were a schema `(dept_name, building, budget)`, then `dept_name` would be a candidate key”
- Denote as a functional dependency:
  - `dept_name → building, budget`
- In `inst_dept`, because `dept_name` is not a candidate key, the building and budget of a department may have to be repeated.
  - This indicates the need to decompose `inst_dept`
- Not all decompositions are good. Suppose we decompose `employee(ID, name, street, city, salary)` into `employee1 (ID, name)`
  `employee2 (name, street, city, salary)`
- The next slide shows how we lose information -- we cannot reconstruct the original `employee` relation -- and so, this is a lossy decomposition.
A Lossy Decomposition

Lossless Join

- Goal: All legal values can be stored in relations
  - Recover originals through join
- Formally: X, Y is a lossless join decomposition of R w.r.t. F if \( \forall r \in R \) satisfying dependencies in F, \( \pi_X(r) \bowtie \pi_Y(r) = r \)
Example of Lossless-Join Decomposition

- **Lossless join decomposition**
- Decomposition of \( R = (A, B, C) \)
  \[ R_1 = (A, B) \quad R_2 = (B, C) \]

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \alpha )</td>
<td>1</td>
<td>A</td>
</tr>
<tr>
<td>( \beta )</td>
<td>2</td>
<td>B</td>
</tr>
</tbody>
</table>

\( r \)

\[ \Pi_{A,B}(r) \quad \Pi_{B,C}(r) \]

\( \Pi_A (r) \Join \Pi_B (r) \)

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \alpha )</td>
<td>1</td>
<td>A</td>
</tr>
<tr>
<td>( \beta )</td>
<td>2</td>
<td>B</td>
</tr>
</tbody>
</table>

Goal — Devise a Theory for the Following

- Decide whether a particular relation \( R \) is in "good" form.
- In the case that a relation \( R \) is not in "good" form, decompose it into a set of relations \( \{ R_1, R_2, ..., R_n \} \) such that
  - each relation is in good form
  - the decomposition is a lossless-join decomposition
- Our theory is based on:
  - functional dependencies
  - multivalued dependencies
Functional Dependencies

- Constraints on the set of legal relations.
- Require that the value for a certain set of attributes determines uniquely the value for another set of attributes.
- A functional dependency is a generalization of the notion of a key.

$X \rightarrow A$ = assertion about a relation $R$ that whenever two tuples agree on all the attributes of $X$, then they must also agree on attribute $A$.

Why do we care?

Knowing functional dependencies provides a formal mechanism to divide up relations (normalization)
- Saves space
- Prevents storing data that violates 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 $r$.

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></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.

---

$K$ is a superkey for relation schema $R$ if and only if $K \rightarrow R$

$K$ is a candidate key for $R$ if and only if

- $K \rightarrow R$, and
- for no $\alpha \subseteq K$, $\alpha \rightarrow R$

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:

$\text{dept\_name} \rightarrow \text{building}$

and $\text{ID} \rightarrow \text{building}$

but would not expect the following to hold:

$\text{dept\_name} \rightarrow \text{salary}$
Use of Functional Dependencies

- 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, by chance, satisfy \( name \rightarrow ID \).

Normalization

Goal = BCNF = Boyce-Codd Normal Form = all FD’s follow from the fact “key \( \rightarrow \) everything.”
- Formally, \( R \) is in BCNF if for every nontrivial FD for \( R \), say \( X \rightarrow A \), then \( X \) is a superkey.
  - “Nontrivial” = right-side attribute not in left side.

Why?
1. Guarantees no redundancy due to FD’s.
2. Guarantees no update anomalies = one occurrence of a fact is updated, not all.
3. Guarantees no deletion anomalies = valid fact is lost when tuple is deleted.
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:

1. $\alpha \rightarrow \beta$ is trivial (i.e., $\beta \subseteq \alpha$)
2. $\alpha$ is a superkey for $R$

Example schema not in BCNF:

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

because $\text{dept_name} \rightarrow \text{building, budget}$ holds on $\text{instr_dept}$, but $\text{dept_name}$ is not a superkey

- Shorthand: combine FD's with common left side by concatenating their right sides.
- Sometimes, several attributes jointly determine another attribute, although neither does by itself. Example:

  Department course_number $\rightarrow$ course_title
Functional Dependencies (Cont.)

- A functional dependency is **trivial** if it is satisfied by all instances of a relation
  - Example:
    - ID, name → ID
    - name → name
  - In general, α → β is trivial if β ⊆ α

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: 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 \( F \).
Example 2

\[
\begin{array}{c|c|c}
\text{Lastname} & \text{Firstname} & \text{Student ID} \\
\hline
\text{Key} & \text{Key} & \text{Superkey} \\
(2 \text{ attributes}) & & \\
\end{array}
\]

Note: There are alternate keys

- **Keys are** \{Lastname, Firstname\} and \{StudentID\}

Who Determines Keys/FD’s?

- **We could assert a key** \(K\).
  - Then the only FD’s asserted are that \(K \rightarrow A\) for every attribute \(A\).
  - No surprise: \(K\) is then the only key for those FD’s, according to the formal definition of “key.”
- **Or, we could assert some FD’s and deduce one or more keys by the formal definition.**
  - E/R diagram implies FD’s by key declarations and many-one relationship declarations.
- **Rule of thumb:** FD’s either come from keyness, many-1 relationship, or from physics.
  - *E.g.*, “no two courses can meet in the same room at the same time” yields room time \(\rightarrow\) course.
Functional-Dependency Theory

- We now consider the formal theory that tells us which functional dependencies are implied logically by a given set of functional dependencies.
- We then develop algorithms to generate lossless decompositions into BCNF and 3NF.
- We then develop algorithms to test if a decomposition is dependency-preserving.

Functional Dependencies (FD’s) and Many-One Relationships

- Consider \( R(A_1,\ldots, A_n) \) and \( X \) is a key then \( X \rightarrow Y \) for any attributes \( Y \) in \( A_1,\ldots, A_n \) even if they overlap with \( X \). Why?
- Suppose \( R \) is used to represent a many \( \rightarrow \) one relationship:
  \( E_1 \) entity set \( \rightarrow E_2 \) entity set
where \( X \) key for \( E_1 \), \( Y \) key for \( E_2 \),
Then, \( X \rightarrow Y \) holds,
And \( Y \rightarrow X \) does not hold unless the relationship is one-one.
- What about many-many relationships?
Inferring FD’s

And this is important because …

- When we talk about improving relational designs, we often need to ask “does this FD hold in this relation?”

Given FD’s $X_1 \rightarrow A_1$, $X_2 \rightarrow A_2$, …, $X_n \rightarrow A_n$, does FD $Y \rightarrow B$ necessarily hold in the same relation?

- Start by assuming two tuples agree in $Y$. Use given FD’s to infer other attributes on which they must agree. If $B$ is among them, then yes, else no.
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 e.g.: 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 $F^+$.

We can find $F^+$, the closure of $F$, by repeatedly 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), and
- complete (generate all functional dependencies that hold).
FDs: Armstrong’s Axioms

- Reflexivity:
  - If \( \{B_1, B_2, \ldots, B_m\} \subseteq \{A_1, A_2, \ldots, A_n\} \Rightarrow A_1A_2\cdots A_n \rightarrow B_1B_2\cdots B_m \)
  - Also called “trivial FDs”

- Augmentation:
  - \( A_1A_2\cdots A_n \rightarrow B_1B_2\cdots B_m \Rightarrow A_1A_2\cdots A_nC_1C_2\cdots C_k \rightarrow B_1B_2\cdots B_mC_1C_2\cdots C_k \)

- Transitivity:
  - \( A_1A_2\cdots A_n \rightarrow B_1B_2\cdots B_m \) and \( B_1B_2\cdots B_m \rightarrow C_1C_2\cdots C_k \Rightarrow A_1A_2\cdots A_n \rightarrow C_1C_2\cdots C_k \)

Armstrong’s Axioms

- 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)

- Owner pet_name age \( \rightarrow \) species
- species \( \rightarrow \) vaccination

- What rule allows us to combine these two FDs?
  A. Reflexivity
  B. Augmentation
  C. Transitivity
  D. Multiple
  E. None
Armstrong’s Axioms

- **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)

- **Owner pet_name age \( \rightarrow \) species**
  - species \( \rightarrow \) vaccination

- **Applying transitivity gives:**
  - A. `pet_name age \( \rightarrow \) species`
  - B. `Owner \( \rightarrow \) vaccination`
  - C. `Vaccination \( \rightarrow \) species`
  - D. `Owner pet_name age \( \rightarrow \) vaccination`
  - E. Transitivity can’t be applied to these rules

---

**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
Algorithm

Define $Y^+ = \text{closure of } Y =$ set of attributes functionally determined by $Y$:

- **Basis:** $Y^+ := Y$.
- **Induction:** If $X \subseteq Y^+$, and $X \rightarrow A$ is a given FD, then add $A$ to $Y^+$.
- **End when $Y^+$ cannot be changed.**

Example

$A \rightarrow B$, $BC \rightarrow D$.

- $A^+ = AB$.
- $C^+ = C$.
- $(AC)^+ = ABCD$. 
Algorithm

- For each set of attributes $X$ compute $X^+$.
  - But skip $X = \emptyset$, $X =$ all attributes.
  - Add $X \rightarrow A$ for each $A$ in $X^+ - X$.
- Drop $XY \rightarrow A$ if $X \rightarrow A$ holds.
  - Consequence: If $X^+$ is all attributes, then there is no point in computing closure of supersets of $X$.
- Finally, project the FD’s by selecting only those FD’s that involve only the attributes of the projection.
  - Notice that after we project the discovered FD’s onto some relation, the eliminated FD’s can be inferred in the projected relation.

Example

$F = AB \rightarrow C, C \rightarrow D, D \rightarrow A$. What FD’s follow?

- $A^+ = A$; $B^+ = B$ (nothing).
- $C^+ = ACD$ (add $C \rightarrow A$).
- $D^+ = AD$ (nothing new).
- $(AB)^+ = ABCD$ (add $AB \rightarrow D$; skip all supersets of $AB$).
- $(BC)^+ = ABCD$ (nothing new; skip all supersets of $BC$).
- $(BD)^+ = ABCD$ (add $BD \rightarrow C$; skip all supersets of $BD$).
- $(AC)^+ = ACD$; $(AD)^+ = AD$; $(CD)^+ = ACD$ (nothing new).
- $(ACD)^+ = ACD$ (nothing new).
- All other sets contain $AB$, $BC$, or $BD$, so skip.
- Thus, the only interesting FD’s that follow from $F$ are: $C \rightarrow A$, $AB \rightarrow D$, $BD \rightarrow C$. 

© 2016 Christopher W. Clifton
Example 2

- Set of FD’s in \( ABCGHI \):
  
  \[
  \begin{align*}
  A & \rightarrow B \\
  A & \rightarrow C \\
  CG & \rightarrow H \\
  CG & \rightarrow I \\
  B & \rightarrow H
  \end{align*}
  \]

- Compute \((CG)^+\), \((BG)^+\), \((AG)^+\)

Example 3

In \( ABC \) with FD’s \( A \rightarrow B, B \rightarrow C \), project onto \( AC \).

1. \( A^+ = ABC \); yields \( A \rightarrow B, A \rightarrow C \).
2. \( B^+ = BC \); yields \( B \rightarrow C \).
3. \( AB^+ = ABC \); yields \( AB \rightarrow C \);
   - drop in favor of \( A \rightarrow C \)
4. \( AC^+ = ABC \) yields \( AC \rightarrow B \);
   - drop in favor of \( A \rightarrow B \)
5. \( C^+ = C \) and \( BC^+ = BC \); adds nothing.

- Resulting FD’s: \( A \rightarrow B, A \rightarrow C, B \rightarrow C \).
- Projection onto \( AC \): \( A \rightarrow C \).
Uses of Attribute Closure

There are several uses of the attribute closure algorithm:

- **Testing for superkey:**
  - To test if \( \alpha \) is a superkey, we compute \( \alpha^+ \) and check if \( \alpha^+ \) contains all attributes of \( R \).

- **Testing functional dependencies**
  - To check if a functional dependency \( \alpha \rightarrow \beta \) holds (or, in other words, is in \( F^+ \)), just check if \( \beta \subseteq \alpha^+ \).
  - That is, we compute \( \alpha^+ \) by using attribute closure, and then check if it contains \( \beta \).
  - Is a simple and cheap test, and very useful

- **Computing closure of \( F \)**
  - For each \( \gamma \subseteq R \), we find the closure \( \gamma^+ \), and for each \( S \subseteq \gamma^+ \), we output a functional dependency \( \gamma \rightarrow S \).

Canonical Cover

- Sets of functional dependencies may have redundant dependencies that can be inferred from the others
  - For example: \( A \rightarrow C \) is redundant in: \( \{ A \rightarrow B, \ B \rightarrow C, A \rightarrow C \} \)
  - Parts of a functional dependency may be redundant
    - E.g.: on RHS: \( \{ A \rightarrow B, \ B \rightarrow C, \ A \rightarrow CD \} \) can be simplified to \( \{ A \rightarrow B, \ B \rightarrow C, \ A \rightarrow D \} \)
    - E.g.: on LHS: \( \{ A \rightarrow B, \ B \rightarrow C, \ AC \rightarrow D \} \) can be simplified to \( \{ A \rightarrow B, \ B \rightarrow C, \ A \rightarrow D \} \)
  - Intuitively, a canonical cover of \( F \) is a "minimal" set of functional dependencies equivalent to \( F \), having no redundant dependencies or redundant parts of dependencies
Extraneous Attributes

Consider a set $F$ of functional dependencies and the functional dependency $\alpha \rightarrow \beta$ in $F$.

- Attribute $A$ is extraneous in $\alpha$ if $A \in \alpha$ and $F$ logically implies $(F - \{\alpha \rightarrow \beta\}) \cup (\{A - A\} \rightarrow \beta)$.
- Attribute $A$ is extraneous in $\beta$ if $A \in \beta$ and the set of functional dependencies $(F - \{\alpha \rightarrow \beta\}) \cup (\{\alpha \rightarrow (\beta - A)\})$ logically implies $F$.

Note: Implication in the opposite direction is trivial in each of the cases above, since a "stronger" functional dependency always implies a weaker one.

Example: Given $F = \{A \rightarrow C, AB \rightarrow C\}$

- $B$ is extraneous in $AB \rightarrow C$ because $\{A \rightarrow C, AB \rightarrow C\}$ logically implies $A \rightarrow C$ (i.e. the result of dropping $B$ from $AB \rightarrow C$).

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 $C$.

Testing if an Attribute is Extraneous

Consider a set $F$ of functional dependencies and the functional dependency $\alpha \rightarrow \beta$ in $F$.

To test if attribute $A \in \alpha$ is extraneous in $\alpha$

1. compute $((\alpha) - A)^+$ using the dependencies in $F$
2. check that $((\alpha) - A)^+$ contains $\beta$; if it does, $A$ is extraneous in $\alpha$

To test if attribute $A \in \beta$ is extraneous in $\beta$

1. compute $\alpha^+$ using only the dependencies in $F' = (F - \{\alpha \rightarrow \beta\}) \cup (\{\alpha \rightarrow (\beta - A)\})$
2. check that $\alpha^+$ contains $A$; if it does, $A$ is extraneous in $\beta$
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.

- To compute a canonical cover for $F$:
  
  repeat
  
  Use the union rule to replace any dependencies in $F$
  
  $\alpha_1 \rightarrow \beta_1$ and $\alpha_1 \rightarrow \beta_2$ with $\alpha_1 \rightarrow \beta_1 \beta_2$

  Find a functional dependency $\alpha \rightarrow \beta$ with an
  extraneous attribute either in $\alpha$ or in $\beta$

  /* Note: test for extraneous attributes done using $F_c$, not $F$*/

  If an extraneous attribute is found, delete it from $\alpha \rightarrow \beta$

  until $F$ does not change

  Note: Union rule may become applicable after some extraneous attributes
  have been deleted, so it has to be re-applied

Computing a Canonical Cover

- $R = (A, B, C)$
- $F = \{A \rightarrow BC, B \rightarrow C, A \rightarrow B, AB \rightarrow C\}$
- Combine $A \rightarrow BC$ and $A \rightarrow B$ into $A \rightarrow BC$
  - Set is now $\{A \rightarrow BC, B \rightarrow C, AB \rightarrow C\}$

  - $A$ is extraneous in $AB \rightarrow C$
  
  - Check if the result of deleting $A$ from $AB \rightarrow C$ is implied by the other
    dependencies
    
    - Yes: in fact, $B \rightarrow C$ is already present!

  - Set is now $\{A \rightarrow BC, B \rightarrow C\}$

  - $C$ is extraneous in $A \rightarrow BC$

  - Check if $A \rightarrow C$ is logically implied by $A \rightarrow B$ and the other dependencies
  
    - Yes: using transitivity on $A \rightarrow B$ and $B \rightarrow C$. $\rightarrow$
      Can use attribute closure of $A$ in more complex cases

  - The canonical cover is: $A \rightarrow B$

  $B \rightarrow C$