Minimal covers

Let's recap

- so far, we discussed **why** it may be necessary to **decompose** a relational schema ***R***, on which a set of functional dependencies ***F*** is defined, in relation to the violation of 3NF (that causes different types of anomalies) and efficiency
- we said that, whatever reason we have for decomposing the schema, the decomposition must satisfy three basic requirements:
  - each sub-schema **must be 3NF**
  - **the** decomposition **must preserve the functional dependencies in F**
  - it must be **possible to reconstruct every legal instance** of the original schema through **the natural join** of the instances of the decomposition

Let's recap

Sapienza
Università di Roma
Dipartimento di Informatica

- we showed how to **verify** that a **given** decomposition (we don't care how it was produced) satisfies all the conditions, in particular we talked about how to verify:
  - if the decomposition preserves the functional dependencies in F
  - whether it is possible to reconstruct every legal instance of the original schema through the natural join of the instances of the decomposition

What's next?

- we now address the problem of **how to** obtain a "good" decomposition
- first of all: **is it always possible to get it?**
- **the answer is YES**: **it is always possible**, given a schema ***R*** on which a set of functional dependencies ***F*** is defined, to **decompose it so as to obtain that**:
  - each sub-schema **is in 3NF**
  - the decomposition **preserves the functional dependencies in F**
  - we **can reconstruct any legal instance** of the original schema through the **natural join** of instances of the decomposition
- **we will present an algorithm that achieves this goal**

Note

Sapienza Università di Roma
Dipartimento di Informatica

- the decomposition that is obtained from the algorithm **is not the only possible one** satisfying the 3 requirements
- **the same algorithm**, depending on its **input** can provide **different** and yet **correct result**
- **<u>there is not just one</u> decomposition,** but there are several possible ones
- so, we cannot use the algorithm to check if a given decomposition is "good", by comparing it with the one provided by the algorithm, as they could be both "good", but different

Before continuing…

Sapienza
Università di Roma
Dipartimento di Informatica

- we introduce the concept of "*minimal cover*" of a set of functional dependencies F
- a minimal cover of F will be the input to the decomposition algorithm
- given a set of functional dependencies ***F***, there can exist **several equivalent minimal covers** **(i.e., having the same closure as *F*)**
- **this is exactly why the decomposition algorithm can produce different correct results**

Definition

Let *F* be a set of functional dependencies.
A **minimal cover** of *F* is a set of functional dependencies G, equivalent to *F*, such that:

- for each functional dependency in *G*, the dependent is a **singleton** (i.e., each **dependent** is **non-redundant**)
- for each dependency $\boldsymbol{X \rightarrow A}$ **in** $\boldsymbol{G}$, there not exists $\boldsymbol{X' \subset X}$ such that $\boldsymbol{G \equiv G - \{X \rightarrow A\} \cup \{X' \rightarrow A\}}$ (i.e., each **determinant** is **non-redundant**)
- there not exists any $\boldsymbol{X \rightarrow A}$ **in** $\boldsymbol{G}$, such that $\boldsymbol{G \equiv G - \{X \rightarrow A\}}$ (i.e., each **dependency** is **non-redundant**)

Comments

SAPIENZA
Università di Roma
Dipartimento di Informatica

- for each functional dependency in *G*, the dependent is a **singleton** (i.e., each **dependent** is **non-redundant**)
  - that is always possible, thanks to the **decomposition** rule

- for each dependency $\boldsymbol{X \rightarrow A}$ **in** $\boldsymbol{G}$, it does not exists a $\boldsymbol{X' \subset X}$ such that $\boldsymbol{G \equiv G - \{X \rightarrow A\} \cup \{X' \rightarrow A\}}$ (i.e., each **determinant** is **non-redundant**)
  - it is not possible to functionally determine A (in G or in G^+) by a **subset of** X

- it does not exist any $\boldsymbol{X \rightarrow A}$ **in** $\boldsymbol{G}$, such that $\boldsymbol{G \equiv G - \{X \rightarrow A\}}$ (i.e., each **dependency** is **non-redundant**)
  - it is not possible to functionally determine A (in G or in G^+) through other dependencies

How it is calculated

For each set of functional dependencies *F* there always exists a minimal cover **equivalent to *F*** that can be obtained in **polynomial** time in three steps:

1. using the decomposition rule, the dependents are reduced to singletons
2. every functional dependency $A_1A_2 \ldots A_{i-1}A_iA_{i+1} \ldots A_n \rightarrow \mathbf{A}$ in F such that $F \equiv F - \{ A_1A_2 \ldots A_{i-1}A_iA_{i+1} \ldots A_n \rightarrow \mathbf{A} \} \cup \{ A_1A_2 \ldots A_{i-1}A_{i+1} \ldots A_n \rightarrow \mathbf{A} \}$ is replaced by $A_1A_2 \ldots A_{i-1}A_{i+1} \ldots A_n \rightarrow \mathbf{A}$; if the latter **already belongs to** F the original dependency is **simply deleted**; the process is repeated **recursively** on $A_1A_2 \ldots A_{i-1}A_{i+1} \ldots A_n \rightarrow \mathbf{A}$; the process ends when **no** functional dependency can be further **reduced**
3. every functional dependency $X \rightarrow A$ in F such that $F \equiv F - \{ X \rightarrow A \}$ is **removed** from F, as it is redundant

Equivalence check

- steps 2 and 3 require to verify the equivalence between two sets of functional dependencies

Equivalence check

- let us recall some definitions and results:
  - $F \equiv G$ if and only if $F^+ = G^+$, i.e., if and only if $F^+ \subseteq G^+$ **and** $G^+ \subseteq F^+$
  - if $F \subseteq G$, trivially $F \subseteq G^+$
  - if $F \subseteq G^+$ then, for the lemma, that $F^+ \subseteq G^+$

- to check whether $F \subseteq G^+$, for each $X \rightarrow Y \in F$ we check whether $X \rightarrow Y \in G^+$, i.e., whether $Y \subseteq (X)^+_G$, i.e., whether Y is in the closure of X, with respect to the set of functional dependencies G (i.e., we can use the algorithm for computing X^+_F)
- **important**: we are still dealing with **the schema R, not yet** decomposed

Equivalence check

SAPIENZA
UNIVERSITÀ DI ROMA
DIPARTIMENTO DI INFORMATICA

- we analyze steps 2 and 3 separately
- we deal with two **particular** cases of the problem of **equivalence between sets of dependencies**

Step 2

SAPIENZA
UNIVERSITÀ DI ROMA
DIPARTIMENTO DI INFORMATICA

- in step 2, **whenever** we want to check the redundancy of an attribute in the determinant of a dependency, we call F the set that contains the original dependency $A_1A_2 \ldots A_{i-1}A_iA_{i+1} \ldots A_n \rightarrow \mathbf{A}$ and G the set that contains the dependency $A_1A_2 \ldots A_{i-1}A_{i+1} \ldots A_n \rightarrow \mathbf{A}$
- so, the two sets differ by **exactly** one dependency, while the remaining ones are equal; so, they **trivially** belong to the closure of both sets
- then, we just have to check that both:
  - $A_1A_2 \ldots A_{i-1}A_iA_{i+1} \ldots A_n \rightarrow \mathbf{A} \in G^+$
  - $A_1A_2 \ldots A_{i-1}A_{i+1} \ldots A_n \rightarrow \mathbf{A} \in F^+$

Step 2

- because of the way the closure algorithm works, we can conclude that **it is not necessary to check** if $A_1A_2...A_{i-1}A_iA_{i+1}...A_n \rightarrow$ **A** $\in G^+$, **that is, if A** $\in$ **(**$A_1A_2...A_{i-1}A_iA_{i+1}...A_n$**)**$^+_G$
- in fact, the algorithm initializes Z to a set that is larger than the one present to the left of the dependency of $A_1A_2...A_{i-1}A_{i+1}...A_n \rightarrow A \in G$, i.e.: $Z = A_1A_2...A_{i-1}A_iA_{i+1}...A_n$
- consequently, because of the assignment $S := \{ A \mid Y \rightarrow V \in G, A \in V \wedge Y \subseteq Z \}$, and in particular of the condition $Y \subseteq Z$, we would immediately insert A in S, because of the presence in **G** of the dependency $A_1A_2...A_{i-1}A_{i+1}...A_n \rightarrow A$
- the theoretical justification for this is in the Armstrong's axioms of reflexivity and transitivity:
- as $A_1A_2...A_{i-1}A_{i+1}...A_n \subseteq A_1A_2...A_{i-1}A_iA_{i+1}...A_n$, by reflexivity we have that $A_1A_2...A_{i-1}A_iA_{i+1}...A_n \rightarrow A_1A_2...A_{i-1}A_{i+1}...A_n$ and, since $A_1A_2...A_{i-1}A_{i+1}...A_n \rightarrow A \in G$, by **transitivity** we obtain that $A_1A_2...A_{i-1}A_iA_{i+1}...A_n \rightarrow A$

Example

given $F = \{AB \rightarrow C, A \rightarrow D, D \rightarrow C\}$

to know if we can eliminate B in $AB \rightarrow C$ we have to check:

1. if $A \rightarrow C \in \mathbf{F^+}$
   - we compute A^+_F and we check if it contains C
2. if $AB \rightarrow C \in \mathbf{G^+}$ with $G = \{A \rightarrow C, A \rightarrow D, D \rightarrow C\}$
   - we compute AB^+_G and we check if it contains C

- point 2 is trivial, as if we compute $(AB)^{\mathbf{+}}_{\mathbf{G}}$ we immediately add C to S and Z, thanks to the dependency $A \rightarrow C$

Step 2

Sapienza Università di Roma
Dipartimento di Informatica

- we still need to check whether $A_1A_2...A_{i-1}A_{i+1}...A_n \rightarrow \mathbf{A} \in F^+$
- we use the algorithm to check if:
    - $\mathbf{A} \in (A_1A_2...A_{i-1}A_{i+1}...A_n)^{+}_{\mathbf{F}}$
- in this case we could be trying to add a constraint that is **not in** the schema definition

- there could be the special case in which the dependency $A_1A_2...A_{i-1}A_{i+1}...A_n \rightarrow A$ is already in F, so it is also in F^+**;** in such a case, **we can directly eliminate the original dependency** (for example, if F contains both $AB \rightarrow C$ and $A \rightarrow C$ then $AB \rightarrow C$ can be eliminated)

- in any other case, if we prove the equivalence of F and G, we can take G as the new reference set **for continuing the reduction** (i.e., the new F is G)

Step 2: observation

SAPIENZA
UNIVERSITÀ DI ROMA
DIPARTIMENTO DI INFORMATICA

- in checking other dependencies, **it is unnecessary to recompute the closures** of groups of attributes for which such a calculation was already performed

- for example, given X, since $(X)^+_F = \{ A \mid X \rightarrow A \in F^+ \}$, and since the **direction** of the check (if $A_1A_2...A_{i-1}A_{i+1}...A_n \rightarrow \mathbf{A} \in F^+$...) implies the computation of closures on a set of dependencies that is either the initial F or a set G that was **already showed to be equivalent to** F (so $F^+ = G^+$), then if $X \rightarrow A \in F^+$ then $X \rightarrow A \in G^+$, i.e., $X^+_F = X^+_G$

Step 2: observation

SAPIENZA
UNIVERSITÀ DI ROMA
DIPARTIMENTO DI INFORMATICA

- if $A_1A_2 \ldots A_{i-1}A_iA_{i+1} \ldots A_n \rightarrow \mathbf{A} \in F$ but **there is no** $Y \rightarrow A \in F$ with $Y \neq A_1A_2 \ldots A_{i-1}A_iA_{i+1} \ldots A_n$, or {subset1 of $A_1A_2 \ldots A_{i-1}A_iA_{i+1} \ldots A_n$} $\rightarrow$ {subset2 of $A_1A_2 \ldots A_{i-1}A_iA_{i+1} \ldots A_n$} $\in F^+$, then it would be **useless to** try to remove attributes to the left of the dependency, since the way the closure is defined, and the way the algorithm works, we won't be able to insert A into any closure, other than the one generated by the dependency $A_1A_2 \ldots A_{i-1}A_iA_{i+1} \ldots A_n \rightarrow \mathbf{A}$ (there are no subsets of $A_1A_2 \ldots A_{i-1}A_iA_{i+1} \ldots A_n$ that determine A, nor combinations $A_1A_2 \ldots A_{i-1}A_iA_{i+1} \ldots A_n \rightarrow Y$ ^ $Y \rightarrow A$ that allow us to apply transitivity, nor {subset1 of $A_1A_2 \ldots A_{i-1}A_iA_{i+1} \ldots A_n$} $\rightarrow$ {subset2 of $A_1A_2 \ldots A_{i-1}A_iA_{i+1} \ldots A_n$} $\in F^+$, which would still allow A to be included in the closure of {subset1 of $A_1A_2 \ldots A_{i-1}A_iA_{i+1} \ldots A_n$}); however, the second condition is not evident, so... better check!
- if $A_1A_2 \ldots A_n \rightarrow A \in F$ and $Y \rightarrow A \in F$ with $Y \subset A_1A_2 \ldots A_n$, we eliminate $A_1A_2 \ldots A_n \rightarrow A$ without checking anything; in fact, in the equivalence check for replacing $A_1A_2 \ldots A_{i-1}A_iA_{i+1} \ldots A_n \rightarrow \mathbf{A}$ with $A_1A_2 \ldots A_{i-1}A_{i+1} \ldots A_n \rightarrow \mathbf{A}$, provided that $Y \subset A_1A_2 \ldots A_{i-1}A_{i+1} \ldots A_n$, it will always be $A \in S$, thanks to the dependency $Y \rightarrow A$

Step 3

- assume that we denote by F the set that contains the dependency $X \rightarrow A$, and by G the set in which this dependency has been eliminated
- again, the two sets **differ by only one dependency**; indeed, it can be verified that **$G \subseteq F$**, so **we already know that $G^+ \subseteq F^+$**
- so, it remains to be checked **that it is $F^+ \subseteq G^+$**, i.e., $F \subseteq G^+$
- **it is enough to check whether $X \rightarrow A \in G^+$**, that is, whether **$A \in X^+_G$**

Step 3: observation

SAPIENZA
UNIVERSITÀ DI ROMA
DIPARTIMENTO DI INFORMATICA

- in this case, however, the closures of groups of attributes **must be recalculated**, because the **direction of** verification leads to the computaion of closures with respect to a set of dependencies **that was not shown to be equivalent to F**

- we also note that if $X \rightarrow A \in F$, but **there is no** $Y \rightarrow A \in F$ with $Y \neq X$, then there **is no point in** trying to eliminate $X \rightarrow A$, since by eliminating this dependency we would no longer be able to determine A

To sum up...

SAPIENZA
UNIVERSITÀ DI ROMA
DIPARTIMENTO DI INFORMATICA

step 2:

- if F isthe set that contains the original dependency $A_1A_2...A_{i-1}A_iA_{i+1}...A_n \rightarrow A$ and G is the set that contains instead the dependency $A_1A_2...A_{i-1}A_{i+1}...A_n \rightarrow A$, to check if $F \equiv G$ it is enough to check:
  if $A_1A_2...A_{i-1}A_{i+1}...A_n \rightarrow A \in F^+$ (so, if $A \in (A_1A_2...A_{i-1}A_{i+1}...A_n)^+_F$)
- if $A_1A_2...A_{i-1}A_{i+1}...A_n \rightarrow A \in F^+$, then we eliminate the dependency $A_1A_2...A_{i-1}A_iA_{i+1}...A_n \rightarrow A$
- if $A_1A_2... A_n \rightarrow A \in F$ and $Y \rightarrow A \in F$ with $Y \subseteq A_1A_2... A_n$, then we eliminate the dependency $A_1A_2... A_n \rightarrow A$
- if $A_1A_2...A_{i-1}A_iA_{i+1}...A_n \rightarrow A \in F$ but **there is no** $Y \rightarrow A \in F$ with $Y \neq A_1A_2...A_{i-1}A_iA_{i+1}...A_n$, then there is no point in trying to remove attributes in the determinant
- **it is not necessary to recalculate the** transitive **closures of** attributes or groups of attributes

To sum up...

step 3:

- F is the set that contains the original dependency $X \rightarrow A$ and G is the set that does not contain it
- to verify if $F \equiv G$ it is enough to verify if $X \rightarrow A \in G^+$ (i.e., $A \in X^+_G$)
- if $X \rightarrow A \in F$ but **there is no** $Y \rightarrow A \in F$ with $Y \neq X$, then it is useless to try to eliminate $X \rightarrow A$
- transitive closures of attributes or attribute groups **must be recalculated**

To sum up...

SAPIENZA
UNIVERSITÀ DI ROMA
DIPARTIMENTO DI INFORMATICA

- there may be **multiple** minimal covers for a given set of functional dependencies.

- using the algorithm, one can **always** find **at least one** minimal cover for **any** set *F*

- it may also be the case that *F* **is already** in minimal form, and so there is **no reduction to be made**