Lossless join

SAPIENZA
UNIVERSITÀ DI ROMA

What does lossless join

usually, a schema is decomposed:

- **when it is not in 3NF**
- to improve efficiency
  - the smaller the size of the tuples, the greater the number of tuples we can load into memory in the same read operation
  - the tuple information is not used entirely by the operations on the database
  - <u>example</u>: the Student schema could be decomposed by separating the personal information (TaxC, Name, Surname, DateB, PlaceB, etc.) from the academic information (Matriculation, Degree, Year, etc.)

Lossless join

Sapienza Università di Roma
Dipartimento di Informatica

- we have seen that when a schema is decomposed, **it is not enough for the** subschemas to be in 3NF

- let's see two examples (one more "abstract", the other more "concrete")

What we want to achieve - 3NF is not enough

- we now consider the schema $R=ABC$ with the set of functional dependencies $F=\{A \rightarrow B, C \rightarrow B\}$ (the schema is not in 3NF due to the presence in F^+ of the partial dependencies $A \rightarrow B$ and $C \rightarrow B$, since the key is AC)
- this schema can be decomposed into:
  - $R1=AB$ with $A \rightarrow B$
  - $R2=BC$ with $C \rightarrow B$
- the resulting schemas, even if they **preserve all dependencies in F^+**, is still not satisfactory.

What we want to achieve - 3NF is not enough

Sapienza Università di Roma
Dipartimento di Informatica

- consider the **legal** instance of *R:*

R

A	B	C
a1	b1	c1
a2	b1	c2

the two facts *(a1,b1,c1)* and *(a2,b1,c2)* are true and not others

- we decompose it by obtaining:

R1

A	B
a1	b1
a2	b1

R2

B	C
b1	c1
b1	c2

What we want to achieve - 3NF is not enough

- if we now compute the natural join we obtain:

R

A	B	C
a1	b1	c1
a2	b1	c2
a1	b1	c2
a2	b1	c1

these tuples did not exist in the original instance!

- **we must ensure that the decomposition and the following natural join does not result in any <u>loss of information</u>**

What we want to achieve - 3NF is not enough

SAPIENZA
UNIVERSITÀ DI ROMA
DIPARTIMENTO DI INFORMATICA

- consider the schema:
- *R={EmployeeID, ProjectID, Manager}* with the set of functional dependencies :
- *F={EmployeeID → ProjectID, ProjectID → Manager*
- a project can have multiple managers but each manager has only one project, and an employee on a project reports to only one manager
- the schema is not in 3NF due to the presence in F^+ of the partial dependencies *EmployeeID → ProjectID* and *ProjectID → Manager*, since the key is (*EmployeeID*, *Manager*)
- the schema can be decomposed into:
  - *R1 = {Matriculation, Project}* with *EmployeeID → ProjectID* and
  - *R2 = {Project, Boss}* with *ProjectID → Manager*
- such a schema, **while preserving all dependencies in F^+**, is not satisfactory

What we want to achieve - 3NF is not enough

SAPIENZA
Università di Roma
Dipartimento di Informatica

- consider the **legal instance** of *R:*

R

EmployeeID	ProjectID	Manager
501	30	E1
502	30	E2

only the two facts *(501,30,E1)* and *(501,30,E2)* are true!

- based on the given decomposition, this instance decomposes into:

R1

EmployeeID	ProjectID
501	30
502	30

R2

ProjectID	Manager
30	E1
30	E2

What we want to achieve - 3NF is not enough

SAPIENZA
UNIVERSITÀ DI ROMA
DIPARTIMENTO DI INFORMATICA

- ...and instead if you join the two legal instances resulting from the decomposition you get

R

EmployeeID	ProjectID	Manager
501	30	E1
502	30	E2
501	30	E2
502	30	E1

tuples unrelated to the reality of interest

What we want to achieve - 3NF is not enough

Sapienza
Università di Roma
Dipartimento di Informatica

in conclusion, the following requirement of the decomposed schema should be met:

- the decomposition must allow us to **reconstruct, by natural join, each legal instance of the original schema** (without adding any extraneous information)

SAPIENZA
UNIVERSITÀ DI ROMA
DIPARTIMENTO DI INFORMATICA

if we decompose a relation schema R we want the obtained **decomposition** $\rho = R_1, R_2, \ldots, R_k$, such that each **legal instance *r* of *R*** can be **reconstructed through natural join from the legal instances $r_1, r_2, \ldots, r_k$ of the decomposition schemas** $R_1, R_2, \ldots, R_k$

as for **reconstructing a tuple *t* of r it is required that $t[R_i] \in r_i$, $\forall\, i = 1,\ldots,k$, then *we* must have $\pi_{R_i}(r) = r_i$, $\forall\, i = 1,\ldots,k$**

Definition

let R be a relation schema; A decomposition $\rho = R_1, R_2, \ldots, R_k$ *has a* **lossless join** if <u>*for each legal instance*</u> ***r*** of R we have

$$r = \pi_{R1}(r) \bowtie \pi_{R2}(r) \bowtie \ldots \bowtie \pi_{Rk}(r)$$

Problem

- again, we start with a given decomposition, and look for a way to verify that it complains with the definition we gave

Theorem

SAPIENZA
UNIVERSITÀ DI ROMA
DIPARTIMENTO DI INFORMATICA

Theorem: let R be a relation scheme and let $\rho = R_1, R_2, \ldots, R_k$ be a decomposition of R; for each legal instance r of R, denoted by $m_\rho(\rho) = \pi_{R1}(r) \bowtie \pi_{R2}(r) \bowtie \ldots \bowtie \pi_{Rk}(r)$, we have:

- $r \subseteq m_\rho(r)$
- $\pi_{Ri}(m_\rho(r)) = \pi_{Ri}(r)$
- $m_\rho(m_\rho(r)) = m_\rho(r)$

Checking lossless join

- given a relation scheme ***R***, a set of functional dependencies ***F***, and a decomposition ρ
- how do we check that the given decomposition has a lossless join?
- there exist an algorithm that checks this in polynomial time
- **we need F as we are implicitly checking legal instances of R**

Algorithm

SAPIENZA
UNIVERSITÀ DI ROMA
DIPARTIMENTO DI INFORMATICA

Algorithm - lossless join

Input: a relation scheme **R**, a set **F** of functional dependencies on R, a decomposition $\rho = R_1, R_2, ..., R_k$

Output: whether or not ρ has a lossless join

begin

construct a table **r** as follows:

r has R columns and k rows

at the intersection of the i-th row and j-th column put:

the symbol a_j **if the attribute** $\mathbf{A \in R_{i,j}}$

the symbol b_{ij} otherwise

repeat

for every $\mathbf{X \rightarrow Y \in F}$

if there are two tuples t_1 and t_2 in r such that $t_1[X]=t_2[X]$ and $t_1[Y] \neq t_2[Y]$

then for every attribute A_j in Y: **if** $t_1[A_j]=a_j$ **then** $t_2[A_j] = t_1[A_j]$

else $t_1[A_j] = t_2[A_j]$

until r has a line with all "a" **or** r has not changed;

if r has a row with all "a" **then:** it has a lossless join

else: it does not have a lossless join

end

a column for each attribute of R and a row for each element of the decomposition (subschema)

the attribute $\boldsymbol{A_j}$ **is part of the subschema** $\boldsymbol{R_i}$

Index i = element of the decomposition = line
Index j = attribute = column

we also correctly handle the case where $t_2[A_j]='a_j'$

even in this case the algorithm <u>always</u> ends! then we need to check if in r there is the tuple we are looking for

Comments

SAPIENZA
UNIVERSITÀ DI ROMA
DIPARTIMENTO DI INFORMATICA

- we can consider the $\mathbf{a_j}$ as particular values belonging to the domain of the attribute $\mathbf{A_j}$
- we can consider $\mathbf{b_{i,j}}$ as particular values belonging to the domain of the attribute $\mathbf{A_j}$
- we can consider **all values** $\mathbf{a_j}$ equal to each other
- the value $\mathbf{b_{ij}}$ is different from $\mathbf{a_j}$ and from another value $\mathbf{b_{kj}}$, even if they all belong to the same domain (that of attribute $\mathbf{A_j}$)
- as a **consequence**, the **initial** r is a particular **instance of** the schema R

Comments

Sapienza Università di Roma
Dipartimento di Informatica

- the algorithm modifies r, so that **all** dependencies in F are **fulfilled**
- whenever it finds two tuples that are equal on the determinant but different on the dependent, it modifies the dependent so they become equal
- in doing so, it prioritizes the "a" symbol ("a" never becomes "b"; "b" can become "a")

Comments

SAPIENZA
UNIVERSITÀ DI ROMA
DIPARTIMENTO DI INFORMATICA

- if two tuples have the same value in the determinant but **different values in the dependent of a** dependency, and only **one of the tuples** has the value "a" in the dependent, we change the "b" of the other tuple into an "a"
- if two tuples have the same value in the determinant but **different values in the dependent of a** dependency, and **neither** has a value of "a" in the dependent, we change one of the tuples so that they have the same value of "b" (e.g., if we have $\mathbf{b_{ij}}$ and $\mathbf{b_{kj}}$ **then we make both values $\mathbf{b_{ij}}$ or $\mathbf{b_{kj}}$**)
- **two values are equal if they are both "*a*" or if they have a "*b*" with the same subscript**
- the algorithm stops when **ALL** pairs of tuples satisfy all the dependencies in F
- so, in the **end, r became a legal instance of R**

Theorem

SAPIENZA
UNIVERSITÀ DI ROMA
DIPARTIMENTO DI INFORMATICA

Theorem: let R be a relation scheme, F a set of functional dependencies on R and let $\rho = R_1, R_2, ..., R_k$ be a decomposition of R; the algorithm correctly decides whether ρ has a lossless join

- **Demonstration:** it must be shown that:
  - has a lossless join ($m_\rho(r)=r$, for each legal instance r)
    - if and only if
- when the algorithm terminates, r has a tuple with all "a"

Theorem: proof

- **if (by contradiction)**
  - suppose **that ρ has a** lossless **join** ($m_\rho(r)= r$) and that when the algorithm terminates the table r does not have any tuple with all "a"
  - *r* is **a legal instance of *R*,** since the algorithm terminates when there **are no more dependency violations in *F***
  - as **no "a" symbol** appearing in the **initial value of r was changed into a "b"** by the algorithm, for each *i*, $i=1,\dots,k$, $\pi_{R_i}(r)$ contains (from the beginning!) a tuple with all *'a'*, which was the one obtained **by projecting the instance *r* on the attributes of R_i**, more precisely in the row corresponding to **the subschema R_i**
  - so, **$m_\rho(r)$ contains a tuple with all *'a'*** and, consequently, **$m_\rho(r)\neq r$ (contradiction)**