Decomposition of relational schemes

Desirable properties of decompositions
Dependency preserving decompositions
Lossless join decompositions

Desirable properties of decompositions 1

Lossless decompositions

A decomposition of the relation scheme R into subschemes R_1, R_2, ..., R_n is **lossless** if, given tuples r_1, r_2, ..., r_n in R_1, R_2, ..., R_n respectively, such that r_i and r_j agree on all common attributes for all pairs of indices (i,j), the – uniquely defined - tuple derived by joining r_1, r_2, ..., r_n is in R.

Terminology: "lossless join" decomposition

Desirable properties of decompositions 3

Dependency preserving decompositions

A decomposition of the relation scheme R into subschemes R_1, R_2, ..., R_n is **dependency preserving** if all the FDs within R can be derived from those within the relations R_1, R_2, ..., R_n.

If F is the set of dependencies defined on R, then the requirement is that the set G of dependencies that can be obtained as projections of dependencies in F^+ onto R_1, R_2, ..., R_n together generate F^+.

Note carefully that it is not enough to check whether projections of dependencies **in F** onto R_1, R_2, ..., R_n together generate F^+.

Desirable properties of decompositions 4

An illustrative example *SCAIP1*

Replace SADDRESS by CITY and AGENT fields in SUPPLIERS(SNAME, SADDRESS, ITEM, PRICE)

Semantics: *Each supplier is based in a city, and the enterprise responsible for setting up the database has an agent for each city*.

Derive in this way a new relation SCAIP(S, C, A, I, P) where S is SNAME, C is CITY, A is AGENT etc.

Desirable properties of decompositions 5

An illustrative example *SCAIP2*

Derive in this way a relation SCAIP(S, C, A, I, P)
where S is SNAME, C is CITY, A is AGENT etc.

The set F of functional dependencies is generated by:

$S \rightarrow C$, $C \rightarrow A$, $S\ I \rightarrow P$

... each supplier sited in one city
... each city has one agent serving it
... each supplier sells each given item at fixed price

Desirable properties of decompositions 6

An illustrative example *SCAIP3*

$F \equiv \{ S \rightarrow C, C \rightarrow A, S\ I \rightarrow P \}$

Consider decomposition { SCA, SIP }:
This is lossless: Suppose that the tuples t_{SCA} and t_{SIP} are in the relations SCA and SIP respectively.

If t_{SCA} and t_{SIP} agree on S, then their join is a tuple $t_{SCAIP} \equiv (s,c,a,i,p)$, where c and a are determined by the attribute s and the i and p attributes are such that p is determined by s and i. Any tuple that satisfies these two FDs is in the relation SCAIP.

Desirable properties of decompositions 7

An illustrative example *SCAIP4*

$F \equiv \{ S \rightarrow C, C \rightarrow A, S\ I \rightarrow P \}$

Consider decomposition { SCA, SIP }:

Also dependency preserving: the sets of dependencies
$\{ S \rightarrow C, C \rightarrow A \}$ and $\{ S\ I \rightarrow P \}$
are included in the projections of F^+ onto SCA and SIP.

This means that the FDs in F, from which all dependencies are generated, are explicit in the sub-schemes SCA and SIP in this case.

Desirable properties of decompositions 8

An illustrative example *SCAIP5*

$F \equiv \{ S \rightarrow C, C \rightarrow A, S\ I \rightarrow P \}$

In decomposition {SIP, SCA}, have problems with SCA.

E.g. update anomaly if want to store an agent for a city in which no supplier is currently located

Get around this by decomposing SCA further:

- decompose as {SC, CA}
- decompose as {SC, SA}
- decompose as {CA, SA}

Desirable properties of decompositions 9

An illustrative example *SCAIP6*

$F \equiv \{ S \rightarrow C, C \rightarrow A, S\ I \rightarrow P \}$

- decompose as { SC, CA }

  this is both lossless join and dependency preserving

- decompose as { SC, SA }

  In this case the images of the FDs in F^+ on SC and SA are $\{S \rightarrow C\}$ and $\{S \rightarrow A\}$ respectively, but the dependency $C \rightarrow A$ can't be inferred. So this decomposition is not dependency preserving.

Desirable properties of decompositions 10

An illustrative example *SCAIP7*

$F \equiv \{ S \rightarrow C, C \rightarrow A, S\ I \rightarrow P \}$

- decompose as {CA, SA}

  In this case, have possibility that Fred is agent for Hull and York, and PVC based in Hull. Then:

  (Hull, Fred) * (PVC, Fred) = (PVC, Hull, Fred)
  (York, Fred) * (PVC, Fred) = (PVC, York, Fred)

  The second join is not in the relation SCA.
  So this decomposition is not lossless join.

Dependency Preserving Decompositions 1

Let R be a relation scheme, ρ a decomposition of R and F a set of functional dependencies of R.

If Z is a set of attributes in R, then

$$\Pi_Z(F) = \{ X \rightarrow Y \in F \mid XY \subseteq Z \}$$

The decomposition ρ is dependency preserving if F is logically implied by the union of the sets of functional dependencies $\Pi_T(F^+)$, where T ranges over all sub-schemes of ρ.

Dependency Preserving Decompositions 2

Illustrative Example

R = ABCD and ρ = {AB, BC, CD}

$F = \{ A \rightarrow B, B \rightarrow C, C \rightarrow D, D \rightarrow A \}$

Question: is ρ dependency preserving?

Certainly $\{ A \rightarrow B, B \rightarrow C, C \rightarrow D \}$ are captured.

How about $D \rightarrow A$? Is also, because

$$F^+ \supseteq \{ B \rightarrow A, C \rightarrow B, D \rightarrow C \}$$

and these FDs are recorded in the sub-schemes

AB, BC, CD.

Hence the dependency $D \rightarrow A$ is also captured.

Dependency Preserving Decompositions 3

Algorithm to check dependency preserving

OK := **true**
for each dependency $X \rightarrow Y$ in F do
begin
  Z := X
  while changes occur in Z do
    for each sub-scheme T of ρ do
      $Z := Z \cup \{ A \mid Z \cap T \rightarrow A \text{ is in } \Pi_T(F^+) \}$
  if not $Z \supseteq Y$ then OK := **false**
end

Dependency Preserving Decompositions 4

Algorithm to check dependency preserving

… while changes occur in Z do
  for each sub-scheme T of ρ do
    $Z := Z \cup \{ A \mid Z \cap T \rightarrow A \text{ is in } \Pi_T(F^+) \}$
...

To compute $\{ A \mid Z \cap T \rightarrow A \text{ is in } \Pi_T(F^+) \}$ calculate

$$((Z \cap T)^+ \cap T)$$

where the closure $(Z \cap T)^+$ is computed with respect to F over the entire relation scheme R.

This avoids need to compute F^+.

Dependency Preserving Decompositions 5

Illustrating the algorithm in action

Consider the relation scheme R = ABCD,
the dependencies $F = \{ A \rightarrow B, B \rightarrow C, C \rightarrow D, D \rightarrow A \}$,
and the decomposition $\rho = \{ AB, BC, CD \}$

Clear that $A \rightarrow B$, $B \rightarrow C$ and $C \rightarrow D$ are preserved
… can prove that the dependency $D \rightarrow A$ is preserved by applying the algorithm

Computation of $\{D\}^+$ over R using F yields {A,B,C,D}

Dependency Preserving Decompositions 6

Illustrating the algorithm in action (cont.)

Computation of $\{D\}^+$ over R using F yields {A,B,C,D}

Z={D} initially. At each iteration of the while-loop, the algorithm introduces a new attribute into Z. For instance, on the first pass, introduce C when T = CD, on second pass, then introduce B when T = BC etc. Hence:
$Z_0 = \{D\}$, $Z_1 = \{C,D\}$, $Z_2 = \{B,C,D\}$, $Z_3 = \{A,B,C,D\}$
where Z_i is the value of Z after the i^{th} iteration.
This proves that dependency $D \rightarrow A$ is preserved.

Lossless Join Decompositions 1

Lossless join decomposition

Let R be a relation scheme, ρ a decomposition of R and F a set of functional dependencies of R. Suppose that the sub-schemes in ρ are $\{R_1, R_2, \ldots, R_k\}$.

ρ has lossless join if every extensional part r for R that satisfies F is such that $r = \Pi_1(r) \,|\times|\, \Pi_2(r) \,|\times|\, \ldots \,|\times|\, \Pi_k(r)$, where $\Pi_i(r)$ denotes the projection of r onto R_i.

Informally: r is the natural join of its projections onto the sub-schemes $R_1, R_2, \ldots, R_k$.

Lossless Join Decompositions 2

Examples (revisited as a reminder)

SCAIP = SIP |×| SCA = SIP |×| SC |×| CA *lossless*

SCA $\subseteq$ SA |×| CA and SCA $\neq$ SA |×| CA *lossy*

… have possibility that Fred is agent for Hull and York, and that PVC is a supplier based in Hull. Then:

(Hull, Fred) * (PVC, Fred) = (PVC, Hull, Fred)
(York, Fred) * (PVC, Fred) = (PVC, York, Fred)

The second join is not in the relation SCA.
So this decomposition is not lossless join.

Lossless Join Decompositions 3

Principles of lossless join decomposition

Let $\rho = \{ R_1, R_2, \ldots, R_k \}$ be a decomposition of R.

Define the mapping $m_\rho( )$ on possible extensions for the relation scheme R [whether or not they satisfy the functional dependencies in R, if there are any], via:

$m_\rho(r) = \Pi_1(r) \,|\times|\, \Pi_2(r) \,|\times|\, \ldots \,|\times|\, \Pi_k(r)$, where $\Pi_i(r)$ denotes the projection of r on sub-scheme R_i.

Notation: use r_i to denote $\Pi_i(r)$, for $1 \Omega i \Omega k$.

Lossless Join Decompositions 4

Principles of lossless join decomposition (cont.)

Lemma: With R, ρ and r_i as above

a) $r \subseteq m_\rho(r)$
b) if $s = m_\rho(r)$, then $\Pi_i(s) = r_i$
c) $m_\rho(m_\rho(r)) = m_\rho(r)$

The condition on $m_\rho()$ specified in part c) identifies it as a **closure** operation.

Cf. *closure* of an interval of real numbers e.g. $1 < \alpha \leq 2$

Lossless Join Decompositions 5

Proof of lemma

a) let $t \in r$. Then $\Pi_i(t) \in r_i$ showing that

$t \in \Pi_1(r) \,|\times|\, \Pi_2(r) \,|\times|\, \ldots \,|\times|\, \Pi_k(r) = m_\rho(r)$

b) by part a) $r \subseteq m_\rho(r)=s$, so that $\Pi_i(s) \supseteq r_i$.
But if $t \in s$, then projection of t onto sub-scheme R_i is in r_i by definition of natural join, so that $\Pi_i(s) \subseteq r_i$ also.

c) $m_\rho(m_\rho(r)) = m_\rho(s)$ by definition of s

$= \Pi_1(s) \,|\times|\, \Pi_2(s) \,|\times|\, \ldots \,|\times|\, \Pi_k(s)$

$= \Pi_1(r) \,|\times|\, \Pi_2(r) \,|\times|\, \ldots \,|\times|\, \Pi_k(r)$

$= m_\rho(r)$ using definition of m_ρ and part b).

Lossless Join Decompositions 6

Testing for lossless join decomposition
assuming all data dependencies in R to be ***functional***

Input: A relation scheme $R=A_1A_2 \ldots A_n$, a set of functional dependencies F, and a decomposition

$\rho = \{ R_1, R_2, \ldots , R_k \}$

Output: ρ is or is not a lossless join decomposition

Construct table of α's and β's, and repeatedly transform the rows by taking account of the FDs until either one row is all α's or no further transformation is possible ...

Lossless Join Decompositions 7

Testing for lossless join decomposition (cont.)

Construct table of α's and β's, and repeatedly transform the rows by taking account of the FDs until either one row is all α's or no further transformation is possible ...

Principle of algorithm: devise a symbolic representation for tuples $s_1, s_2, \ldots , s_k$ from $R_1, R_2, \ldots , R_k$ respectively that are joinable, and for tuples $t_1, t_2, \ldots , t_k$ in R so that s_i is projection of t_i onto R_i for each i. Impose all those conditions on $t_1, t_2, \ldots , t_k$ that follow from the FDs in F. If none of the t_i's is the join of $s_1, s_2, \ldots , s_k$, then they define an extension for R that exhibits a lossy join.

Lossless Join Decompositions 8

Testing for lossless join decomposition (cont.)

Construct table of α's and β's, and repeatedly transform the rows by taking account of the FDs until either one row is all α's or no further transformation is possible ...

Principle of algorithm: devise a symbolic representation for tuples $s_1, s_2, \ldots , s_k$ from $R_1, R_2, \ldots , R_k$ respectively that are joinable, and for tuples $t_1, t_2, \ldots , t_k$ in R so that s_i is projection of t_i onto R_i for each i. Impose all those conditions on $t_1, t_2, \ldots , t_k$ that follow from the FDs in F. If none of the t_i's is the join of $s_1, s_2, \ldots , s_k$, then they define an extension for R that exhibits a lossy join.

Lossless Join Decompositions 9

Method of testing for lossless join decomposition

1. Construct a table
   with n columns (corresponding to attributes)
   with k rows (corresponding to sub-schemes)

   Initialise the table at row i column j
   by entering α_j if attribute A_j appears in R_i
   and by entering β_{ij} otherwise

NB α's represent joinable tuples, padded out to R by β's

Lossless Join Decompositions 10

Method of testing for lossless join decomposition (cont.)

2. Repeatedly modify the table to take account of all dependencies until no further updates occur

   i.e. if $X \rightarrow Y$ and two rows agree on all the attributes in X then modify them so that they also agree on all attributes in Y. Explicitly, change attributes in Y thus:

   - if one symbol is an α_i make the other an α_i
   - if both symbols are of form β_{*j} make both β_{ij} or $\beta_{i'j}$ arbitrarily.

   On termination declare lossless join if and only if one of the rows is $\alpha_1\alpha_2 \ldots \alpha_n$.

Lossless Join Decompositions 11

Illustrative example

Verify the decomposition SCAIP = SIP |×| SC |×| CA is a lossless join ....

Initial table

	S	C	A	I	P
SIP	α_1	β_{12}	β_{13}	α_4	α_5
SC	α_1	α_2	β_{23}	β_{24}	β_{25}
CA	β_{31}	α_2	α_3	β_{34}	β_{35}

Functional dependencies are $S \rightarrow C$, $C \rightarrow A$, $S\ I \rightarrow P$

Lossless Join Decompositions 12

Illustrative example

Functional dependencies are $S \rightarrow C$, $C \rightarrow A$, $S\ I \rightarrow P$ and from these arrive via stage 2 of algorithm at table:

	S	C	A	I	P
SIP	α_1	$\boldsymbol{\alpha_2}$	$\boldsymbol{\alpha_3}$	α_4	α_5
SC	α_1	α_2	α_3	β_{24}	β_{25}
CA	β_{31}	α_2	$\boldsymbol{\alpha_3}$	β_{34}	β_{35}

at which point no further dependencies apply.

Row 1 shows that the result is lossless

Lossless Join Decompositions 13

Principle of the lossless join algorithm illustrated ...

Consider the example: in the initial table

	S	C	A	I	P
SIP	α_1	β_{12}	β_{13}	α_4	α_5
SC	α_1	α_2	β_{23}	β_{24}	β_{25}
CA	β_{31}	α_2	α_3	β_{34}	β_{35}

… the rows can be seen as representing generic tuples from SIP, SC and CA that are joinable (i.e. agree on all common attributes). The join of these three tuples will necessarily be $\alpha_1\alpha_2\alpha_3\alpha_4\alpha_5$.

Lossless Join Decompositions 14

Principle of the lossless join algorithm illustrated ...

Key question: are the functional dependencies enough to ensure that $\alpha_1\alpha_2\alpha_3\alpha_4\alpha_5$ is itself a tuple in the relation SCAIP?

After modification to take account of all FDs, suitable tuples matching the template for equality of values in the 3 rows in the table define a valid extensional part for SCAIP: can substitute them to get a concrete relation r.

Either one of the 3 tuples *is* $\alpha_1\alpha_2\alpha_3\alpha_4\alpha_5$ *lossless*

or $\alpha_1\alpha_2\alpha_3\alpha_4\alpha_5 \in m_\rho(r) \setminus r$ *lossy*

Lossless Join Decompositions 15

Algorithm shows that SCA is a lossy join of SA and CA:

FDs are S→C, C→A		S	C	A
initial and	SA	α_1	β_{12}	α_3
final form of table	CA	β_{21}	α_2	α_3

Fred is agent for Hull [β_{12}] and York, PVC is based in Hull, there is another supplier [β_{21}] say GPT at York.

Take as extension of SCA the pair of *valid* tuples:

(PVC, Hull, Fred) [row 1] and (GPT, York, Fred) [row 2]

Project onto SA and CA, get

(PVC, Fred), (Hull, Fred), (GPT, Fred), (York, Fred)

Take natural join to get *rogue* tuples:

(PVC, York, Fred) [$\alpha_1\,\alpha_2\,\alpha_3$], (GPT, Hull, Fred) [$\beta_{21}\,\beta_{12}\,\alpha_3$]

Lossless Join Decompositions 16

Theorem

If ρ = {S, T} is a decomposition of R, and F is the set of FDs for R, then ρ is a lossless join decomposition with respect to F if and only if

either $T\setminus S \subseteq (S \cap T)^+$ or $S\setminus T \subseteq (S \cap T)^+$.

Proof: Applying the method of the algorithm to test for lossless join, get initial table of the form:

	S ∩ T	S\T	T\S
T	α...α	β...β	α...α
S	α...α	α...α	β...β

Lossless Join Decompositions 17

Theorem

If $\rho = \{S, T\}$ is a decomposition of R, and F is the set of FDs for R, then ρ is a lossless join decomposition with respect to F if and only if either $S\backslash T \subseteq (S \cap T)^+$ or $S\backslash T \subseteq (S \cap T)^+$.

Proof (cont.) ... get initial table of the form:

	$S \cap T$	$S\backslash T$	$T\backslash S$
T	$\alpha...\alpha$	$\beta...\beta$	$\alpha...\alpha$
S	$\alpha...\alpha$	$\alpha...\alpha$	$\beta...\beta$

The final table is this table modified so that every column labelled by an attribute in $(S \cap T)^+$ is changed to an α, from which the theorem follows.

Lossless Join Decompositions 18

Application of Thm: SCA is a lossy join of SA and CA, as neither of the dependencies $A \rightarrow S$, $A \rightarrow C$ is valid.

Corollary to the theorem: If R is a relation scheme, and $X \rightarrow A$ is a functional dependency in R, where A is a an attribute, X is a set of attributes not containing A, and XA is a proper subset of R, then $R_1=XA$, $R_2=R\backslash A$ is a lossless join decomposition of R.

Proof: $R_1 \cap R_2 \supseteq X$, hence $R_1\backslash R_2 = A \in (R_1 \cap R_2)^+$.

Lossless Join Decompositions 19

Exercise for lossless join algorithm from Ullman 1982:

Take R = ABCDE

$R_1 = AD$, $R_2 = AB$, $R_3 = BE$, $R_4 = CDE$, $R_5 = AE$

with the functional dependencies

$A \rightarrow C$, $B \rightarrow C$, $C \rightarrow D$, $DE \rightarrow C$, $CE \rightarrow A$

In this example, the identification of β_{*j}'s is crucial.

Can trace the algorithm through three stages …

Lossless Join Decompositions 20

Split R = ABCDE into $R_1=AD$, $R_2=AB$, $R_3=BE$, $R_4=CDE$, $R_5=AE$ with the FDs $A \rightarrow C$, $B \rightarrow C$, $C \rightarrow D$, $DE \rightarrow C$, $CE \rightarrow A$

	A	B	C	D	E
AD	α_1	β_{12}	β_{13}	α_4	β_{15}
AB	α_1	α_2	β_{23}	β_{24}	β_{25}
BE	β_{31}	α_2	β_{33}	β_{34}	α_5
CDE	β_{41}	β_{42}	α_3	α_4	α_5
AE	α_1	β_{52}	β_{53}	β_{54}	α_5

Lossless Join Decompositions 21

Split R = ABCDE into R_1=AD, R_2=AB, R_3=BE, R_4=CDE, R_5=AE
with the FDs A → C, B → C, C → D, DE → C, CE → A

	A	B	C	D	E
AD	α_1	β_{12}	$\boldsymbol{\beta_{13}}$	α_4	β_{15}
AB	α_1	α_2	$\boldsymbol{\beta_{13}}$	β_{24}	β_{25}
BE	β_{31}	α_2	$\boldsymbol{\beta_{13}}$	β_{34}	α_5
CDE	β_{41}	β_{42}	α_3	α_4	α_5
AE	α_1	β_{52}	$\boldsymbol{\beta_{13}}$	β_{54}	α_5

Lossless Join Decompositions 22

Split R = ABCDE into R_1=AD, R_2=AB, R_3=BE, R_4=CDE, R_5=AE
with the FDs A → C, B → C, C → D, DE → C, CE → A

	A	B	C	D	E
AD	α_1	β_{12}	$\boldsymbol{\alpha_3}$	α_4	β_{15}
AB	α_1	α_2	$\boldsymbol{\alpha_3}$	$\boldsymbol{\alpha_4}$	β_{25}
BE	$\boldsymbol{\alpha_1}$	α_2	$\boldsymbol{\alpha_3}$	$\boldsymbol{\alpha_4}$	α_5
CDE	$\boldsymbol{\alpha_1}$	β_{42}	α_3	α_4	α_5
AE	α_1	β_{52}	$\boldsymbol{\alpha_3}$	$\boldsymbol{\alpha_4}$	α_5