Decomposition of a Relation

Decomposition is the process of breaking down in parts or elements. It replaces a relation with a collection of smaller relations. It breaks the table into multiple tables in a database.

Decomposition is used to eliminate some of the problems of bad design like anomalies, inconsistencies, and redundancy.

Test the decmposition is lossless or lossy

Figure shows the algorithm for Test the decmposition is lossless or lossy

Testing Lossless Joins

It turns out to be fairly easy to tell whether a decomposition has a lossless join with respect to a set of functional dependencies.

Algorithm 7.2: Testing for a Lossless Join.
Input: A relation scheme $R = A_1 \cdots A_n$, a set of functional dependencies F, and a decomposition $\rho = (R_1, \ldots, R_k)$.
Output: A decision whether ρ is a decomposition with a lossless join.
Method: We construct a table with n columns and k rows; column j corresponds to attribute A_j, and row i corresponds to relation scheme R_i. In row i and column j put the symbol a_j if A_j is in R_i. If not, put the symbol b_{ij} there.

Repeatedly "consider" each of the dependencies $X \rightarrow Y$ in F, until no more changes can be made to the table. Each time we "consider" $X \rightarrow Y$, we look for rows that agree in all the columns for the attributes of X. If we find two such rows, equate the symbols of those rows for the attributes of Y. When we equate two symbols, if one of them is a_j, make the other be a_j. If they are b_{ij} and $b_{\ell j}$, make them both b_{ij} or $b_{\ell j}$, arbitrarily.

If after modifying the rows of the table as above, we discover that some row has become $a_1 \cdots a_k$, then the join is lossless. If not, the join is lossy (not lossless). $\square$

Lossless Join Example discussed in class

Let R = ABCDE, R1 = AD, R2 = AB, R3 = BE, R4 = CDE, and R5 = AE. Let the functional dependencies be: A -> C, B -> C, C -> D, DE -> C, CE -> A
Apply algorithm 7.2 from class handout to test if the decomposition of R into {R1,..,R5} is a lossless join decomposition.

The initial table looks as follows:

	A	B	C	D	E
R1(AD)	**a1**	b12	b13	**a4**	b15
R2(AB)	**a1**	**a2**	b23	b24	b25
R3(BE)	b31	**a2**	b33	b34	**a5**
R4(CDE)	b41	b42	**a3**	**a4**	**a5**
R5(AE)	**a1**	b52	b53	b54	**a5**

Apply FD A -> C to the initial table to modify the violating dependencies. Rows 1, 2, 5 will need to change the values for the RHS attribute C – equate b13, b23, b53 to b13 (you might very well have picked b23 or b53 to equate all three). We won't change the rows 3 & 4 yet because their symbols b31, b41 are different from a1.

A	B	C	D	E
a1	b12	~~b13~~ **b13**	a4	b15
a1	a2	~~b23~~ **b13**	b24	b25
b31	a2	b33	b34	a5
b41	b42	a3	a4	a5
a1	b52	~~b53~~ **b13**	b54	a5

Apply B -> C next to equate b33 with b13

A	B	C	D	E
a1	b12	~~b13~~ b13	a4	b15
a1	**a2**	~~b23~~ **b13**	b24	b25
b31	**a2**	~~b33~~ **b13**	b34	a5
b41	b42	a3	a4	a5
a1	b52	~~b53~~ b13	b54	a5

Next, use C -> D to equate a4, b24, b34, and b54

A	B	C	D	E
a1	b12	~~b13~~ **b13**	**a4**	b15
a1	a2	~~b23~~ **b13**	~~b24~~ **a4**	b25
b31	a2	~~b33~~ **b13**	~~b34~~ **a4**	a5
b41	b42	a3	a4	a5
a1	b52	~~b53~~ **b13**	~~b54~~ **a4**	a5

DE -> C helps us to equate b13 (all occurrences) with a3. The following table shows the corresponding changes.

A	B	C	D	E
a1	b12	~~b13~~ ~~b13~~ **a3**	a4	b15
a1	a2	~~b23~~ ~~b13~~ **a3**	~~b24~~ a4	b25
b31	a2	~~b33~~ ~~b13~~ **a3**	~~b34~~ **a4**	**a5**
b41	b42	**a3**	**a4**	**a5**
a1	b52	~~b53~~ ~~b13~~ **a3**	~~b54~~ **a4**	**a5**

Apply CE -> A to the table above to equate b31, b41, and a1.

A	B	C	D	E
a1	b12	~~b13~~ ~~b13~~ a3	a4	b15
a1	a2	~~b23~~ ~~b13~~ a3	~~b24~~ a4	b25
~~b31~~ **a1**	**a2**	~~b33~~ ~~b13~~ **a3**	~~b34~~ **a4**	**a5**
~~b41~~ **a1**	b42	**a3**	a4	**a5**
a1	b52	~~b53~~ ~~b13~~ **a3**	~~b54~~ a4	**a5**

The middle row in the table above is all a's, and the decomposition has a lossless join.

What happens if we apply FDs in a different order?

Try any FD with the same symbols (a or b) on the LHS attribute in at least two rows. We have two choices: A -> B or B -> C. We tried A -> B in the previous test. Try B -> C here.

A	B	C	D	E
a1	b12	b13	**a4**	b15
a1	**a2**	~~b23~~ b23	b24	b25
b31	**a2**	~~b33~~ b23	b34	**a5**
b41	b42	**a3**	**a4**	**a5**
a1	b52	b53	b54	**a5**

Apply A -> C after that to get the following table. Notice I chose b23 to equate b13, b23, and b53.

A	B	C	D	E
a1	b12	~~b13~~ b23	**a4**	b15
a1	**a2**	~~b23~~ b23	b24	b25
b31	**a2**	~~b33~~ b23	b34	**a5**
b41	b42	**a3**	**a4**	**a5**
a1	b52	~~b53~~ b23	b54	**a5**

The table shown above is the similar to the one we got before after applying A -> C and B-> C to the initial table. To the current table we apply C -> D as before because we have the same symbol b23 in rows 1, 2, 3, and 5.