Fall 2007 CS186 Discussion Section: Week 12, 11/12 - 11/16

Your friendly TAs

November 13, 2007

1 Attribute Closure

Consider the attribute set R = ABCDE and the FD set F = {AB→C, A→D, D→E, and AC→B}. Compute the attribute closure for each of A, AB, B, and D.

- A: {A, D, E}
- AB: {A, B, C, D, E} (Since the closure of AB contains all the attributes, AB is a key of the relation).
- B: {B}
- D: {D, E}

2 Schema Decomposition

Decompose the following attribute sets, R, and FD sets, F, into (a) BCNF and (b) 3NF:

- **R = ABCEG; F = {AB→C, AC→B, BC→A, E→G}.**

  AB→C violates BCNF. So, decompose R into ABEG and ABC. Now, ABC is in BCNF. ABEG is not, however due to the FD E→G. So, decompose into ABE and EG. Both of these are in BCNF. The final solution: ABC, ABE, EG.

  To convert BCNF decomposition into dependency-preserving 3NF, we need to first find an FD from the minimal cover that violates dependency-preservation. In this case, F is already a minimal cover and there are no FDs that span multiple relations. So, the BCNF solution above is dependency-preserving.

- **R = ABCDE, F = {AB→C, DE→C, and B→D}.**

  AB→C violates BCNF. So, decompose R into ABDE and ABC. ABC is BCNF. ABDE is not, due to the B→D FD. So, decompose ABDE into ABE and BD. BD is in BCNF (every 2 attribute relation is). So is ABE. Final solution: ABC, ABE, BD.

  F is already a minimal cover. There is one FD that violates dependency preservation: DE→C. We solve this problem by adding the relation DEC. Final solution: ABE, BD, DEC, ABC.

- **R = ABCDEFG, F= {AB→CD, C→EF, G→A, G→F, CE→F}.**

  AB→CD violates BCNF. Decompose into ABEFG and ABCD. ABCD is now BCNF (AB is a key). The FD G→A violates BCNF for ABEFG. Decompose into BEFG and GA. GA is in BCNF. BEFG is not due to G→F. Decompose into BEG and GF. Final solution: ABCD, GA, BEG, GF. Note that a better solution that is also BCNF would result if we first combined (union) the two FDs G→A, G→F, creating G→AF. Then the solution would be: ABCD, BEG, GAF.

  F is not a minimal cover. First, put FDs in standard form using the decomposition axiom: {AB→C, AB→D, C→E, C→F, G→A, G→F, CE→F}. Second, eliminate unnecessary attributes from the left-hand side. Since C→E, CE→F can be reduced to C→F: {AB→C, AB→D, C→E, C→F, G→A, G→F, C→F}. Finally, remove redundant FDs, i.e. those FDs that are not necessary to compute the closure of F. In this case, we remove C→F. The minimal cover is {AB→C, AB→D, C→E, C→F, G→A, G→F}.

  The two FDs, C→E, C→F, cannot be checked by looking at a single relation (a join would be necessary). Thus they violate dependency preservation. We could fix this by adding two relations CE and CF, so that the dependencies did not require a join to check. *As an optimization*, we could first use the union axiom to derive C→EF and then add the single relation CEF. Final solution: ABCD, BEG, GAF, CEF.

- **R = ABCDEFGH, F = {ABH→C, A→DE, BGH→F, F→ADH, BH→GE}.**

  Violating FD: A→DE. Decomposition: ABCFGH and ADE. For ADE, no violating FD; it is in BCNF. For ABCFGH, the following FDs of F+ exist: {ABH→C, BGH→F, F→AH, BH→G}. Note that the last two FDs were derived from those given in F using the (unfortunately named) decomposition axiom. F→AH violates BCNF. Decomposition: BCFG, FAH. Both are in BCNF. Final solution: ADE, BCFEG, FAH.

  The minimal cover of F is {ABH→C, A→D, A→E, BH→F, F→A, F→D, F→H, BH→G, BH→E}. The following FDs violate dependency preservation: {ABH→C, F→D, BH→F , BH→G, BH→E}. We can optimize these using the union axiom (reduces the number of relations created for dependency preservation while still being 3NF): {ABH→C, F→D, BH→GEF}. Thus, to obtain dependency preservation, we need to add relations ABHC, FD, and BHGEF to those above.

3 Lossless? Dependency Preserving?

Which of the following decompositions of R = ABCDEG, with F = {AB→C, AC→B, AD→E, B→D, BC→A, E→G}is (i) dependency-preserving? (ii) lossless-join?

- **AB, BC, ABDE, EG**

  The decomposition {AB, BC, ABDE, EG} is not lossless. To prove this consider the following instance of R: {(a1, b, c1, d1, e1, g1), (a2, b, c2, d2, e2, g2)}. Because of the functional dependencies BC→A and AB→C, a1≠a2 if and only if c1≠c2. It is easy to see that the join of AB and BC contains four tuples: {(a1, b, c1), (a1, b, c2), (a2, b, c1), (a2, b, c2)} So the join of AB, BC, ABDE and EG will contain at least 4 tuples, (actually it contains 8 tuples) so we have a lossy decomposition here. This decomposition does not preserve the FD, AB→C (or AC B).

- **ABC, ACDE, ADG**

  The decomposition {ABC, ACDE, ADG} is lossless. Intuitively, this is because the join of ABC, ACDE, and ADG can be constructed in two steps; first construct the join of ABC and ACDE: this is lossless because their (attribute) intersection is AC which is a key for ABC, so this is lossless. Now join this intermediate join (ABCDE) with ADG. This is also lossless because the attribute intersection is AD and AD→ADG. So by the test mentioned in the text this step is also a lossless decomposition.

  The project of the FDs of R onto ABC gives us: AB→C, AC→B and BC→A. The projection of the FDs of R onto ACDE gives us: AD→E and the projection of the FDs of R onto ADG gives us: AD→G (by transitivity). The closure of this set of dependencies does not contain E→G nor does it contain B→D. So this decomposition is not dependency preserving.