CS 186 Discussion Section
Week 12

Peter Alvaro and Kuang Chen

April 20, 2009

Review ER Design elements

- Entities and weak entities (rectangles)
- Relationships (diamonds)
- Arrows: many-to-many, one-to-one, one-to-many
- Bold lines: at least one

Functional Dependencies

Decomposition motivation

Want to avoid anomalies caused by redundancy:

- update anomalies
- insertion anomalies
- deletion anomalies

Problems arise when where is a forced association between attributes
To remedy that, we want to decompose a relation into smaller relations

Example

Functional dependencies in $Address(street_address, city, state, zip)$?
$street\_address, city, state \rightarrow zip$
$zip \rightarrow city, state$
$zip, state \rightarrow zip$ (trivial FD, $LHS \supseteq RHS$)
$zip \rightarrow state, zip$

Review FD Closure and Armstrong's Axioms

Recall:

- Closure of a set of FDs
- Attribute closure with respect to a set of FDs

Armstrong's Axioms:

- Reflexivity: if $X \supseteq Y$, then $X \rightarrow Y$
- Augmentation: if $X \rightarrow Y$, then $XZ \rightarrow YZ$ for any Z
- Transitivity: if $X \rightarrow Y$ and $Y \rightarrow Z$, then $X \rightarrow Z$

Also infer:

- Union: if $X \rightarrow Y$ and $X \rightarrow Z$, then $X \rightarrow YZ$
- Decomposition: if $X \rightarrow YZ$, then $X \rightarrow Y$ and $X \rightarrow Z$

FD Exercises

Flights schema:

```
Flights(flight_no, date, from, to, plane_id) PK(flight_no, date), FK(plane_id)
denoted FDRTP
Planes(plane_id, type) PK(plane_id)
denoted PY
Seat(seat_no, legroom, plane_id) PK(seat_no, plane_id), FK(plane_id)
denoted SLP
```

1. Find the set of functional dependencies

   From the key constraints we get the following functional dependencies:
   $DF \rightarrow DFRTP$
   $P \rightarrow PY$
   $SP \rightarrow SPL$
   Also, because a flight always flies between a given pair of airports, we get:
   $F \rightarrow RT$

2. Expand the FDs found above using Armstrongs axioms (you can omit the trivial and non interesting dependencies).

   We can obtain additional functional dependencies using Armstrongs axioms. Using decomposition and transitivity ($DF \rightarrow DFRTP$, $P \rightarrow PY$) we can obtain:
   $DF \rightarrow Y$
   Using decomposition, augmentation and transitivity ($DF \rightarrow DFRTP$, $SP \rightarrow SPL$) we can obtain:
   $DFS \rightarrow L$

3. You are given two additional conditions as follows:

   (a) Each flight has a Boolean attribute has_first_class

   (b) Each seat has an additional attribute *seat_color*. Seat colors match the exterior color of the planes.

   Can you modify the answers to question 1 above using the results obtained here?

   The two additional conditions lead to more functional dependencies. The additional attribute has $first\_class$ *is denoted as H and the seat_color as C. The presence of first class in a flight is a function of the type of the plane, so in other words H is functionally determined by Y. Thus we get the condition $Y \rightarrow H$, we know that $P \rightarrow Y$ so by transitivity:*
   $P \rightarrow H$
   Since the color of the seat matches the exterior of the plane,

$P \rightarrow C$
The last two functional dependencies can be used to modify the relational representation; since C is functionally determined by P, we make seat_color an attribute of Plane. We now have an attribute that depends on type, which is not a key of Planes. Therefore, we should create a new table $Plane\_type(type, has\_first\_class)$ *to prevent redundancies (and therefore avoid anomalies).*

Closure Exercises

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

- 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: $\{D, E\}$

Review Decomposition

- Need to balance the need to decompose to avoid redundancy problems with potential decomposition problems
- Normal forms allow us to reason about what redundancy issues may arise
- Decomposition properties:
  - lossless-join: able to recover any instance of the original relation from the corresponding instances of the smaller relations
    if every instance r of R that satisfies the dependencies, $\pi_X(r) \bowtie \pi_Y(r) = r$
  - dependency preservation: able to enforce constraints on the original relation by just enforcing constraints on the smaller relations
    if $(F_X \cup F_Y)^+ = F^+$
- Decomposing a relation may require joins to answer queries

Normal Forms

Each are increasingly restrictive, 1NF and 2NF aren't much talked about, but:

- 1NF: every attribute has an atomic value, i.e. no sets or lists
- 2NF: a non-key attribute cannot be functionally dependent on a proper subset of a candidate key (partial dependencies not allowed)

Let R be the set of attributes in a relation, $X \subseteq R$, and A an attribute of R
Relation is in 3NF if for every FD $X \rightarrow A$ one of the following are true:

1. $A \in X$ (trivial FD), or
2. X is a superkey, or
3. A is part of some key for R (also called being "prime")

For BCNF, only first two properties are options.

Decomposing

- Into BCNF:
  If $X \to A$ is in FD and violates BCNF, then decompose R into $R - A$ and XA
  Repeat if necessary.

- Above will be lossless-join decomposition. Need more for 3NF to be dependency-preserving:
  Recall *minimal cover* of a FD is equivalent set of dependencies such that: (1) each LHS is necessary and each RHS is single attribute, and (2) ever dependency is required for F^+. Construct minimal cover by:

  1. Put FDs in standard form (single attribute on RHS)
  2. Minimize LHS of each FD (remove it if can preserve F^+)
  3. Delete redundant FDs

  For 3NF: Starting with lossless-join decomposition $R_1 \ldots R_n$ and a minimal cover of FDs F. Let F_i be the projection of F on R_i.

  1. Identify the set of dependences N in F that are not included in the closure of the $F_1 \cup F_2 \cdots \cup F_n$
  2. For each FD $X \to A$ in N, create a relation schema XA and add it to the decomposition

Decomposition Exercises

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.

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.

Transactions

(see attached)