

Theory behind Normalization & DB Design

Lecture 12

Satisfiability: Does an FD hold?

- Satisfiability of FDs
- Given: FD $X \rightarrow Y$ and relation R
- Output: Does R satisfy $X \rightarrow Y$?
- Algorithm:
 - a. Sort R on X
 - b. Do all the tuples with equal X values agree on their Y values?

| Original Database | | | | |
|-------------------|-----------|-------------|----------|--------|
| St-Name | Status | Course-Name | Course-# | Salary |
| Ben | senior | db | 670 | 500 |
| Ben | senior | ds | 680 | 500 |
| Dan | freshment | db | 670 | 400 |
| Dan | Junior | ds | 680 | 600 |

| <p>$\{St-Name, Status\} \rightarrow \{Salary\}$? YES:</p> <table style="width: 100%; border-collapse: collapse;"> <thead> <tr> <th>St-Name</th> <th>Status</th> <th>Salary</th> </tr> </thead> <tbody> <tr> <td>Ben</td> <td>senior</td> <td>500</td> </tr> <tr> <td>Ben</td> <td>senior</td> <td>500</td> </tr> </tbody> </table> <table style="width: 100%; border-collapse: collapse;"> <tbody> <tr> <td>Dan</td> <td>freshment</td> <td>400</td> </tr> </tbody> </table> <table style="width: 100%; border-collapse: collapse;"> <tbody> <tr> <td>Dan</td> <td>Junior</td> <td>600</td> </tr> </tbody> </table> | St-Name | Status | Salary | Ben | senior | 500 | Ben | senior | 500 | Dan | freshment | 400 | Dan | Junior | 600 | <p>$\{St-Name, Status\} \rightarrow \{Course-\#\}$? NO:</p> <table style="width: 100%; border-collapse: collapse;"> <thead> <tr> <th>St-Name</th> <th>Status</th> <th>Course-#</th> </tr> </thead> <tbody> <tr> <td>Ben</td> <td>senior</td> <td>670</td> </tr> <tr> <td>Ben</td> <td>senior</td> <td>680</td> </tr> </tbody> </table> <table style="width: 100%; border-collapse: collapse;"> <tbody> <tr> <td>Dan</td> <td>freshment</td> <td>670</td> </tr> </tbody> </table> <table style="width: 100%; border-collapse: collapse;"> <tbody> <tr> <td>Dan</td> <td>Junior</td> <td>680</td> </tr> </tbody> </table> | St-Name | Status | Course-# | Ben | senior | 670 | Ben | senior | 680 | Dan | freshment | 670 | Dan | Junior | 680 |
|--|-----------|----------|--------|-----|--------|-----|-----|--------|-----|-----|-----------|-----|-----|--------|-----|--|---------|--------|----------|-----|--------|-----|-----|--------|-----|-----|-----------|-----|-----|--------|-----|
| St-Name | Status | Salary | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| Ben | senior | 500 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| Ben | senior | 500 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| Dan | freshment | 400 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| Dan | Junior | 600 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| St-Name | Status | Course-# | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| Ben | senior | 670 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| Ben | senior | 680 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| Dan | freshment | 670 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| Dan | Junior | 680 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |

Checking for above functional dependencies

Inference of FD's

- The set of all FDs on R-a brute force algorithm:
 - Find all possible candidate FDs on attributes of R
 - For each candidate apply the satisfiability algorithm
 - Time consuming!
- Alternative: Inference
 - Some FDs may be inferred from other FDs
 - given: {St-Name, Course-#} → Salary
 - implies: {St-Name, Course-#, Status} → Salary
- Basically
 - Start from some set F of FDs
 - Derive all possible FDs from F

Based on Rules of Inference

- The set of all FDs derivable from F is called the **closure F+** of F

| | | |
|-------|--|--------------|
| F | a. (St-Name) → Status b. (St-Name, Course-#) → Salary | |
| F^+ | 1. (St-Name) → St-Name 2. (Course-#, Status) → (Course-#, Status) 3. (Course-#, Status) → Status | Reflexive |
| ... | | |
| | i. From (a): (St-Name, Course-#) → (Status, Course-#) | Augmentation |
| ... | | |

Armstrong's Rules of Inference

- Reflexive (Intuition: superset implies subset (need not be proper)
 - (Name, Sex) → Name
 - (Name, Sex) → (Name, Sex)
- Transitivity
 - $X \rightarrow Y, Y \rightarrow Z$ implies $X \rightarrow Z$
- Augmentation (augment both sides of the implication)
 - (Name, Sex) → Name implies
 - (Name, Sex, Age) → (Name, Age)
- **Rules are sound and complete**
 - Produces only FD's in the closure
 - Produces all FD's in closure

Algorithm: Computing FD closures

- The closure F^+ of F is the set of FD's inferred from F .
- Algorithm:
 - $F^+ := F$
 - repeat
 - apply Armstrong's inference rules on F^+
 - $F^+ := F$
 - Until F^+ is not augmented
 - end

Determining closure of X (attributes) under F (set of FD's)

- Given F :
 - 1. $AB \rightarrow E$
 - 2. $BE \rightarrow I$
 - 3. $E \rightarrow C$
 - 4. $CI \rightarrow D$
- Derive $\{A, B\}^+$:
 - a. $\{A, B, E\}$ (given X)
 - b. $\{A, B, E, I\}$ (FD #2 in F)
 - c. $\{A, B, E, I, C\}$ (FD #3 in F)
 - d. $\{A, B, E, I, C, D\}$ (FD #4 in F)

Requirements on Decompositions

- **Lossless Decomposition**
- Sometimes not possible
 - A B C D E
 - Given the following fd's
 - $A \rightarrow BCDE$
 - $CD \rightarrow E$
 - $CE \rightarrow B$
 - 3NF would dictate
 - Either $\{A, B, C, D\}$ & $\{C, D, E\}$
 - Or $\{A, B, C, E\}$ & $\{C, E, B\}$
 - In either case you lose a functional dependency

Dependency Preserving 3NF

- Objective: Introduce a dependency-preserving decomposition algorithm 3NF
- The subset F_X of the closure F^+ which uses only attributes of X is the projection of F on X
- A decomposition of R into $R(X)$ and $R(Y)$ is dependency preserving if $F^+ = (F_X \cup F_Y)^+$
- Trick is to use the minimal cover of F to drive decomposition

Minimal Cover

- F is a minimal set of FDs if each $X \rightarrow Y$ is
 - $F = \{A \rightarrow BE, AB \rightarrow DE, AC \rightarrow G\}$
 - Canonical: $|Y| = 1$ (use decomposition)
 - $A \rightarrow B, A \rightarrow E, AB \rightarrow D, AB \rightarrow E, AC \rightarrow G$
 - Left-reduced: X can't be replaced by a subset
 - $A \rightarrow B, A \rightarrow E, AC \rightarrow G$
 - $A \rightarrow D$ (How?)
 - Non-redundant: $X \rightarrow Y$ can't be removed

Dependency-Preserving 3NF Decomposition Algorithm

- Find minimal cover
- Put FDs agreeing on the left-hand-side in the same schema
- Have extra schema for unaccounted attributes
- Example
 - $R = \{A, B, C, D, E, G, I, J\}$
 - $A \rightarrow B$
 - $A \rightarrow E$
 - $A \rightarrow D$
 - $AC \rightarrow G$
 - Resulting Schemas

– A B D E A C G I J

NOTE THIS IS NOT THE DECOMPOSITION YOU WOULD GET USING 3NF NORMALIZATION, BUT THIS IS IN 3NF

Lossless Joins

- A Decomposition is a set of projections of relational schemas
- A natural join should return the original table

| R | A | B | C | $\pi_{A,B}$ | A | B | $\pi_{B,C}$ | B | C |
|---|----|----|----|-------------|----|----|-------------|----|----|
| | a1 | b1 | c1 | | a1 | b1 | | b1 | c1 |
| | a2 | b2 | c2 | | a2 | b2 | | b2 | c2 |
| | a3 | b1 | c3 | | a3 | b1 | | b1 | c3 |

Not Lossless!

| $\pi_{A,B}(R) * \pi_{B,C}(R)$ | A | B | C |
|-------------------------------|----|----|----|
| | a1 | b1 | c1 |
| | a2 | b2 | c2 |
| | a3 | b1 | c3 |
| | a1 | b1 | c3 |
| | a3 | b1 | c1 |

Lossless decomposition

| R | A | B | C | $\pi_{A,B}$ | A | B | $\pi_{B,C}$ | B | C |
|---|----|----|----|-------------|----|----|-------------|----|----|
| | a1 | b1 | c1 | | a1 | b1 | | b1 | c1 |
| | a2 | b2 | c2 | | a2 | b2 | | b2 | c2 |
| | a3 | b1 | c1 | | a3 | b1 | | | |

$$\pi_{A,B}(R) \cap \pi_{B,C}(R) = B$$

R1, R2 is a lossless join decomposition of R iff the attributes common to R1 and R2 contain a key for at least one of the involved relations

Dependency-Preserving Lossless-Join 3NF Decomposition Algorithm

- Find minimal cover
- Put FDs agreeing on the left-hand-side in the same schema
- Have extra schema for a key, if none of the above schemas contain a key
- $R = \{A, B, C, D, E, G, I, J\}$
- $F = \{A \rightarrow B, A \rightarrow E, A \rightarrow D, AC \rightarrow G\}$
- Resulting Schemas:
 - A B E D
 - A C G
 - A C I J

NOTE THIS IS THE DECOMPOSITION YOU WOULD GET USING 3NF NORMALIZATION.

Facts

- Schemas can always employ lossless-join dependency-preserving decompositions to achieve 3NF
- Determining whether a relationship schema satisfies 3NF is NP-complete. Hence, automated normalization is tough.
- Not all violations to BCNF can be resolved through dependency-preserving decompositions
 - MANAGER PROJECT DEPT
 - $\{PROJECT, DEPT\} \rightarrow MANAGER$
 - $MANAGER \rightarrow DEPT$
- Doesn't satisfy BCNF but is in 3NF
- BCNF decompositions can't always preserve dependencies
