Lossless Decomposition

A decomposition is lossless if we can recover:

R(A, B, C)

Decompose

R1(A, B) R2(A, C)

Recover

R'(A, B, C)

Thus, R' = R

Testing for Losslessness

- A (binary) decomposition of **R** = (R, **F**) into **R**1= (R1, **F**1) and **R**2 = (R2, **F**2) is lossless if and only if:
  - either the FD $(R1 \cap R2) \rightarrow R1$ is in **F**+
  - or the FD $(R1 \cap R2) \rightarrow R2$ is in **F**+

  - all attributes common to both R1 and R2 functionally determine ALL the attributes in R1 OR
  - all attributes common to both R1 and R2 functionally determine ALL the attributes in R2

Projecting FDs

- Given:
  - a relation *R*
  - the set *F* of FDs that hold in *R*
  - a relation $R_i \subset R$
- Determine the set of all FDs F_i that
  - Follow from *F* and
  - Involve only attributes of R_i

FD Projection Algorithm

- Start with $F_i = \emptyset$
- For each subset X of R_i
  - Compute X^+
  - For each attribute A in X^+
    - If A is in R_i
      - add X -> A to F_i
- Compute the minimal basis of F_i

Making projection more efficient

- Ignore trivial dependencies
  - No need to add X -> A if A is in X itself
- Ignore trivial subsets
  - The empty set or the set of all attributes (both are subsets of X)
- Ignore supersets of X if $X^+ = R$
  - They can only give us "weaker" FDs (with more on the LHS)

Example: Projecting FDs

- *Given R(A,B,C)* with FDs *A->B* and *B->C*
  - A^+=*ABC* ; yields *A->B, A->C*
    - We ignore A->A as trivial
    - We ignore the supersets of A, AB^+ and AC^+, because they can only give us "weaker" FDs (with more on the LHS)
  - B^+=*BC* ; yields *B->C*
  - C^+=*C* ; yields nothing.

Example cont'd

- Resulting FDs: *A->B*, *A->C*, and *B->C*
- Projection onto *AC* : *A->C*
  - Only FD that involves a subset of {*A*,*C*}
- Projection on BC: *B->C*
  - Only FD that involves subset of {B, C}

Projection is expensive

- Even with these tricks, projection is still expensive.
- Suppose R_1 has n attributes.
  How many subsets of R_1 are there?

$$2^n - 1$$

Part III:
Normal Forms

Database Design Theory

- General idea:
  - Express constraints on the data
  - Use these to decompose the relations
- Ultimately, get a schema that is in a “normal form” that guarantees good properties, such as no anomalies.
- “Normal” in the sense of conforming to a standard.
- The process of converting a schema to a normal form is called normalization.

Acknowledgements: M. Papagelis, R. Johnson

Motivation for normal forms

- Identify a “good” schema
  - For some definition of “good”
  - Avoid anomalies, redundancy, etc.
- Many normal forms
  - 1^{st}
  - 2^{nd}
  - 3^{rd}
  - Boyce-Codd
  - ... and several more we won’t discuss...

$$BCNF \subseteq 3NF \subseteq 2NF \subseteq 1NF$$

1st Normal Form (1NF)

- No multi-valued attributes allowed
  - Imagine storing a list of values in an attribute
- Counter example
  - Course(name, instructor, [student,email]*)

Name	Instructor	Student Name	Student Email
CS 3DB3	Chiang	Alice	alice@gmail
		Mary	mary@mac
		Mary	mary@mac
SE 3SH3	Miller	Nilesh	nilesh@gmail

2nd normal form (2NF)

- Non-key attributes depend on candidate keys
  - Consider non-key attribute A
  - Then there exists an FD X s.t. X -> A, and X is a candidate key
- Counter-example
  - Movies(title, year, star, studio, studioAddress, salary)
  - FD: title, year -> studio; studio -> studioAddress; star->salary

Title	Year	Star	Studio	StudioAddr	Salary
Star Wars	1977	Hamill	Lucasfilm	1 Lucas Way	$100,000
Star Wars	1977	Ford	Lucasfilm	1 Lucas Way	$100,000
Star Wars	1977	Fisher	Lucasfilm	1 Lucas Way	$100,000
Patriot Games	1992	Ford	Paramount	Cloud 9	$2,000,000
Last Crusade	1989	Ford	Lucasfilm	1 Lucas Way	$1,000,000

3rd normal form (3NF)

- Non-prime attr. depend *only* on candidate keys
  - Consider FD X -> A
  - Either X is a superkey OR A is *prime* (part of a key)
- Counter-example:
  - studio -> studioAddr

    (*studioAddr* depends on *studio* which is not a candidate key)

Title	Year	Studio	StudioAddr
Star Wars	1977	Lucasfilm	1 Lucas Way
Patriot Games	1992	Paramount	Cloud 9
Last Crusade	1989	Lucasfilm	1 Lucas Way

3NF, dependencies, and join loss

- Theorem: always possible to convert a schema to lossless join, dependency-preserving 3NF
- Caveat: always *possible* to create schemas in 3NF for which these properties do not hold
- FD loss example 1:
  - MovieInfo(title, year, studioName)
  - StudioAddress(title, year, studioAddress)

  => Cannot enforce studioName -> studioAddress
- Join loss example 2:
  - Movies(title, year, star)
  - StarSalary(star, salary)

  => Movies $\bowtie$ StarSalary yields additional tuples

Boyce-Codd normal form (BCNF)

- One additional restriction over 3NF
  - All non-trivial FDs have superkey LHS
- Counterexample
  - CanadianAddress(street, city, province, postalCode)
  - Candidate keys: {street, postalCode}, {street, city, province}
  - FD: postalCode -> city, province

  - Satisfies 3NF: city, province both prime
  - Violates BCNF: postalCode is not a superkey

=> Possible anomalies involving postalCode