53

Part II:
Schema Decomposition

53

54

Relational Schema Design

- Goal of relational schema design is to avoid redundancy, and the anomalies it enables.
  - *Update anomaly* : one occurrence of a fact is changed, but not all occurrences have been updated.
  - *Deletion anomaly* : valid fact is lost when a tuple is deleted.

54

55

Result of bad design: Anomalies

name	addr	beersLiked	manf	favBeer
Janeway	Voyager	Bud	A.B.	WickedAle
Janeway	Voyager	WickedAle	Pete's	WickedAle
Spock	Enterprise	Bud	A.B.	Bud

- Update anomaly: if Janeway is transferred to *Intrepid*, will we remember to change each of her tuples?
- Deletion anomaly: If nobody likes Bud, we lose track of the fact that Anheuser-Busch manufactures Bud.

55

56

Example of Bad Design

Suppose we have FDs name -> addr, favBeer and beersLiked -> manf. This design is bad:

Drinkers(<u>name</u>, addr, <u>beersLiked</u>, manf, favBeer)

name	addr	beersLiked	manf	favBeer
Janeway	Voyager	Bud	A.B.	WickedAle
Janeway	???	WickedAle	Pete's	???
Spock	Enterprise	Bud	???	Bud

Data is redundant, because each of the ???'s can be figured out by using the FDs.

56

Goal of Decomposition

57

- Eliminate redundancy by decomposing a relation into several relations
- Check that a decomposition does not lead to bad design

57

FDs and redundancy

58

Given relation R and FDs F

- R often exhibits anomalies due to redundancy
- F identifies many (not all) of the underlying problems

Idea

- Use F to identify "good" ways to split relations
- Split R into 2+ smaller relations having less redundancy
- Split F into subsets which apply to the new relations (compute the projection of functional dependencies)

58

Schema decomposition

59

- Given relation R and FDs F
  - Split R into R_i s.t. for all i $R_i \subset R$ (no new attributes)
  - Split F into F_i s.t. for all i, F entails F_i (no new FDs)
  - F_i involves only attributes in R_i
- Caveat: entirely possible to lose information
  - F^+ may entail FD f which is not in $(U_i\ F_i)^+$

  => Decomposition lost some FDs
  - Possible to have $R \subset \bowtie_i R_i$

  => Decomposition lost some relationships
- Goal: minimize anomalies without losing info

59

Good Properties of Decomposition

60

1) Lossless Join Decomposition
   - When we join decomposed relations we should get ***exactly*** what we started with
2) Avoid anomalies
   - Avoid redundant data
3) Dependency Preservation
   - $(F_1 \cup \ldots \cup F_n)^+ = F^+$

60

Problem with Decomposition

61

Given instances of the decomposed relations, we may not be able to reconstruct the corresponding instance of the original relation – information loss

61

Example: Splitting Relations

62

Student_Name	Student_Email	Course	Instructor
Alice	alice@gmail	SE 4DB3	Chiang
Alice	alice@gmail	CS 3SH3	Zheng
Bob	bob@mcmaster	SE 3RA3	Janicki
Laura	laura@gmail	SE 4DB3	Jones

Students (email, name)

Courses (code, instructor)

Taking (email, courseCode)

Students ⋈ Taking ⋈ Courses has additional tuples!

- (Alice, alice@gmail, SE4DB3, Jones), but Alice is not in Jones' section of SE 4DB3
- (Laura, laura@gmail, SE4DB3, Chiang), but Laura is not in Chiang's section of SE 4DB3

Why did this happen? How to prevent it?

62

Information loss with decomposition

63

- Decompose R into S and T
  - Consider FD A->B, with A only in S and B only in T
- FD loss
  - Attributes A and B no longer in same relation

  => Must join T and S to enforce A->B (expensive)
- Join loss
  - Neither (S ∩ T) -> S nor (S ∩ T) -> T in F^+

  => Joining T and S produces extraneous tuples

63

Lossless Join Decomposition

64

- A decomposition should not lose information
- A decomposition ($\mathbf{R}_1, \ldots, \mathbf{R}_n$) of a schema, **R**, is **lossless** if every valid instance, r, of **R** can be reconstructed from its components:
  - $r = r_1 \bowtie \ldots \bowtie r_n$ where $r_i = \Pi_{R_i}(r)$

64

Lossy Decomposition

65

r

ID	Name	Addr
11	Pat	1 Main
12	Jen	2 Pine
13	Jen	3 Oak

$r_1 = \Pi_{R1}(r)$

ID	Name
11	Pat
12	Jen
13	Jen

$r_2 = \Pi_{R2}(r)$

Name	Addr
Pat	1 Main
Jen	2 Pine
Jen	3 Oak

$r_1 \bowtie r_2$

ID	Name	Addr
11	Pat	1 Main
12	Jen	2 Pine
13	Jen	3 Oak
12	Jen	3 Oak
13	Jen	2 Pine

65

What is lost?

66

- Lossy decomposition
  - Loses the fact that (12, Jen) lives at 2 Pine (not 3 Oak)
  - Loses the fact that (13, Jen) lives at 3 Oak
- Remember: lossy decompositions yield ***more*** tuples than they should when relations are joined together

r

ID	Name	Addr
11	Pat	1 Main
12	Jen	2 Pine
13	Jen	3 Oak

$r_1 = \Pi_{R1}(r)$

ID	Name
11	Pat
12	Jen
13	Jen

$r_2 = \Pi_{R2}(r)$

Name	Addr
Pat	1 Main
Jen	2 Pine
Jen	3 Oak

ID	Name	Addr
11	Pat	1 Main
12	Jen	2 Pine
13	Jen	3 Oak
12	Jen	3 Oak
13	Jen	2 Pine

66

Example 2

67

R

Model Name	Price	Category
a11	100	Canon
s20	200	Nikon
a70	150	Canon

R1

Model Name	Category
a11	Canon
s20	Nikon
a70	Canon

R2

Price	Category
100	Canon
200	Nikon
150	Canon

Ack: S.M. Lee

67

Example 2 (cont'd)

68

R1 $\bowtie$ R2

Model Name	Price	Category
a11	100	Canon
a11	150	Canon
s20	200	Nikon
a70	100	Canon
a70	150	Canon

R

Model Name	Price	Category
a11	100	Canon
s20	200	Nikon
a70	150	Canon

68

Lossy decomposition

69

- Additional tuples are obtained along with original tuples
- Although there are more tuples, this leads to less information
- Due to the loss of information, the decomposition for the previous example is called lossy decomposition or lossy-join decomposition

69

Testing for Losslessness

70

- 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 ∩ R2) → R1 is in **F**+
  - or the FD (R1 ∩ R2) → 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

70

Decomposition Property

71

- In our example
  - *Name* $\not\rightarrow$ *ID,Name*
  - *Name* $\not\rightarrow$ *Name, Addr*
- *A lossless decomposition*
  - *[ID,Name] and [ID,Addr]*
- Example 2:
  - *Category* $\not\rightarrow$ ModelName, Category
  - *Category* $\not\rightarrow$ Price, Category
  - Better to use [MN, Category] and [MN, Price]
- In other words, if R1 ∩ R2 forms a superkey of either R1 or R2, the decomposition of R is a lossless decomposition

71

Lossless Decomposition

72

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

72

Example : Lossless Decomposition

73

Given:

Lending-schema = (*branch-name, branch-city, assets, customer-name, loan-number, amount*)

FDs:

branch-name $\longrightarrow$ *branch-city, assets*

loan-number $\longrightarrow$ *amount, branch-name*

Decompose Lending-schema into two schemas:

Branch-schema = (*branch-name, branch-city, assets*)

Loan-info-schema = (*branch-name, customer-name, loan-number, amount*)

73

Example : Lossless Decomposition

74

Show that the decomposition is a Lossless Decomposition

Branch-schema = (*branch-name, branch-city, assets*)

Loan-info-schema = (*branch-name, customer-name, loan-number, amount*)

- Since *Branch-schema* ∩ *Loan-info-schema* = {*branch-name*}
- We are given: *branch-name* $\longrightarrow$ *branch-city, assets*

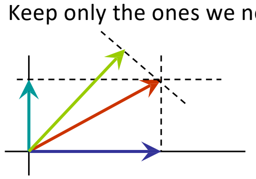
Thus, this decomposition is lossless.

74

Projecting FDs

75

- Once we've split a relation, we have to re-factor our FDs to match
  - Each FDs must only mention attributes from one relation
- Similar to geometric projection
  - Many possible projections (depends on how we slice it)
  - Keep only the ones we need (minimal basis)



75

Projecting FDs

76

- 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

76

FD Projection Algorithm

77

- 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

77

Making projection more efficient

78

- 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)

78

Example: Projecting FDs

79

- *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.

79

Example cont'd

80

- 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}

80

Projection is expensive

81

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

$$2^n - 1$$

81