COS 597D:
Principles of
Database and Information Systems

Managing Functional Dependencies and Redundancy

General functional constraints
(Review)

General form for relational model:
- Let α(R) denote the set of names of attributes in the schema for relation R
- Let X and Y be subsets of α(R)

The functional dependency $X \rightarrow Y$ holds if for any instance I of R and for any pair of tuples t_1 and t_2 of R,

$$\pi_X(t_1) = \pi_X(t_2) \Rightarrow \pi_Y(t_1) = \pi_Y(t_2)$$

- special cases: candidate keys, superkeys

Functional Constraint in SQL

Why store state?

```
CREATE TABLE  Student
   ( sid  CHAR(10),
     street CHAR(40),
     city CHAR(40)
     state CHAR(40)
     zipcode CHAR(10)
     PRIMARY KEY  (sid)
     CHECK  (not exist
                ( SELECT  *
                  FROM  Student S
                  WHERE S.zipcode=zipcode
                     AND S.state<>state)
                )
      )
```

Redundancy

- Functional dependencies capture redundancy in a relation
  e.g. zipcode→ state: why store state?
- Redundancy good for reliability
- Redundancy bad for
  – space to store: repetitions
  – must maintain on changes
  – representation of one relationship embedded in another

Example relation for a city elementary school system:
school_child: (<u>name, st_addr, apt.</u>, birthday, school)
st_addr → school
consider a large apt. building

Solution: decompose

Example:
child: (<u>name, st_addr, apt.</u>, birthday)
placement: (<u>st_addr</u>, school)

- child ◊◊ placement gives school_child
  because of *functional dependency*
- space gain larger than space cost
- functional dependency now primary key constraint
- st_addr, school correspondence explicitly maintained
- computation cost?

General Form:
- for X, Y ⊆ α(R) and X → Y
- decompose R into
  R1: α(R) - (Y-X)
  R2: X U Y

More problematic decompose

Example:
school_child: (<u>school, stuID</u>, st_addr, apt., birthday)
st_addr → school

becomes
stu: (stuID, st_addr, apt., birthday)
placement: (<u>st_addr</u>, school)

General Form:
for X, Y ⊆ α(R)
and X → Y
decompose R into
•R1: α(R) - (Y-X)
•R2: X U Y

Constraint (school, stuID) → (st_addr, apt., birthday)
- was primary key constraint
- now split constraint
  to check requires ◊◊ - expensive
- primary key for stu?

Primary key for example

Example:

school_child: (school, stuID, st_addr, apt., birthday)
st_addr → school

becomes stu: (stuID, st_addr, apt., birthday)
placement: (st_addr, school)

primary key for stu?

(stuID, st_addr) → (stuID, st_addr, school)
(stuID, st_addr, school) → (stuID, st_addr, apt., birthday)
so **stu: (stuID, st_addr, apt., birthday)**

★ new primary key constraint does not imply old primary key constraint:
(school, stuID) → (st_addr, apt., birthday)

Propose: decompose to eliminate redundancy

Two examples

1. school_child: (name, st_addr, apt., birthday, school)
   st_addr → school
   ↓
   child: (name, st_addr, apt., birthday)
   placement: (st_addr, school)

2. school_child: (school, stuID, st_addr, apt., birthday)
   st_addr → school
   ↓
   stu: (stuID, st_addr, apt., birthday)
   placement: (st_addr, school)
   (school, stuID) → (st_addr, apt., birthday)

Decomposition: Formal Properties

- Let Φ be a set of functional dependencies (FDs) for a relational scheme R with attribute set $\alpha(R)$
- Let Φ^+ denote the set of all FDs implied by Φ
  the closure of Φ
- Let X, Y $\subseteq \alpha(R)$, where $X \cap Y$ is not necessarily empty
- Let Φ_x denote set of FDs V→W in Φ^+ with $V \subseteq X$ and $W \subseteq X$
- Decomposition of R into R_1: X and R_2:Y is
  - **lossless** if for every instance I of R that satisfies Φ
    $\pi_X(I)$ ◊◊ $\pi_Y(I) = I$
    - guaranteed to get back R
  - **dependency preserving** if $(\Phi_x \cup \Phi_Y)^+ = \Phi^+$
    - can check all FDs for R by checking all for X and all for Y without doing JOIN

Implied functional dependencies

- Definition: a functional dependency X→Y is implied by Φ if X→Y holds whenever all functional dependencies in Φ hold
- Armstrong's Axioms
  for attribute sets X, Y, Z
  1. if $X \subseteq Y$ then $Y \to X$ reflexivity
  2. if $X \to Y$ then $\forall Z\ (XZ \to YZ)$ augmentation
  3. if $X \to Y$ and $Y \to Z$ then $X \to Z$ transitivity
- Theorem: The set of all functional dependences obtained from Φ by repeated application of Armstrong's Axioms gives Φ^+

Normal Forms

- How do we find "good" ("best"?) decomposition?
- Identify normal forms with desirable properties
  - must be lossless – can't lose anything
  - should be dependency preserving - avoid need for joins to check dependencies
- Decompose so resulting relations are in normal form

Boyce-Codd Normal Form (BCNF)

- Let R denote a relational scheme with attribute set $\alpha(R)$
- R is in BCNF with respect to a set Φ of FDs if for all FDs in Φ^+ of the form X→Y with X, Y $\subseteq \alpha(R)$, at least one of
  - $Y \subseteq X$ (trivial func. dep.)
  - X is a superkey for R
- very strong normal form
- can't always get dependency preserving decomposition into set of BCNF relations

Third Normal Form (3NF)

- Let R denote a relational scheme with attribute set α(R)
- R is in 3NF with respect to a set Φ of FDs if for all FDs in Φ^+ of the form X→Y with X, Y ⊆ α(R), at least one of
  - Y ⊆ X (trivial func. dep.)
  - X is a superkey for R
  - each attribute A in Y-X is contained in a candidate key for R
- can always get lossless, dependency preserving decomposition into 3NF relations
- cannot always remove all functional dependencies

Why allow right hand side part of some candidate key?

- consider decomposing R using X → A
  - A an attribute
  - X not superkey
  - A not in X
- get R_1: α(R) - (A) and R_2: X U {A}
- if A not part of a candidate key then

  for any candidate key K ⊆ α(R)

  check K → α(R) -{A} in R_1

  including K → X

  check X → A in R_2

  conclude K → A

  all checks local to R_1 or R_2

  NO HARM DECOMPOSE
- if A is part of a candidate key K

  splitting key: K-A in α(R_1); K ∩ (X U {A}) in α(R_2)

  to check K is a candidate key need R_1 ◊◊ R_2 AVOIDING

Revisit example

Lossless-join decomposition?
Dependency preserving decomposition?
Normal forms?

school_child: (school, stuID, st_addr, apt., birthday)

st_addr → school

becomes

stu: (stuID, st_addr, apt., birthday)

placement: (st_addr, school)

Constraint (school, stuID) → (st_addr, apt., birthday)

- was primary key constraint
- now split constraint

  to check requires ◊◊ - expensive

Decompositon to achieve 3NF

- Is polynomial-time algorithm for 3NF lossless-join, dependency-preserving decomposition
- Can require adding "extra" relation.
- Get at expense of redundancy

Example

R with attributes ABCD; AB primary key;

other functional dependencies A→C; B→C

decompose R1: ABD; R2: BC

lossless? dependency-preserving?

Decompositon to achieve 3NF

- Is polynomial-time algorithm for 3NF lossless dependency-preserving decomposition
- Can require adding "extra" relation.
- Get at expense of redundancy

Example

R with attributes ABCD; AB primary key; A→C; B→C

decompose R1: ABD; R2: BC

add R3: AC

redundant because can get from R1 ◊◊ R2

Discussion

- Consider normal forms when designing relations.
- Using 3NF minimizes problems of general functional dependencies
  - does not eliminate
- Use BCNF if can get it
  - decomposition algorithm simpler too!