Intro to DB

CHAPTER 7
RELATIONAL DB DESIGN

Chapter 7: Relational Database Design

- Features of Good Relational Design
- Atomic Domains and First Normal Form
- Decomposition Using Functional Dependencies
- Functional Dependency Theory
- Algorithms
- Decomposition Using Multivalued Dependencies
- More Normal Form
- Database-Design Process
- Modeling Temporal Data

Pitfalls of Relational Database Design

- Relational database design

  Find a "good" collection of relation schemas for our information need

  - R = (A B C D E) <----- single relation schema
  - $DB_1 = \{ R_1, \ldots\ldots, R_n \}$ <----- DB schema (set of relation schemas)
- Design Goals:
  - Ensure that relationships among attributes are represented (information content)
  - Avoid redundant data
  - Facilitate enforcement of database integrity constraints
- A bad design may lead to
  - Inability to represent certain information
  - Repetition of Information
  - Loss of information

Example

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

branch-name	*branch-city*	*assets*	*customer-name*	*loan-number*	*amount*
Downtown	Brooklyn	9000000	Jones	L-17	1000
Redwood	Palo Alto	2100000	Smith	L-23	2000
Perryridge	Horseneck	1700000	Hayes	L-15	1500
Downtown	Brooklyn	9000000	Jackson	L-14	1500

- Redundancy:
  - Data for *branch-name, branch-city, assets* are repeated for each loan that a branch makes
  - Wastes space
  - Complicates updating, introducing possibility of inconsistency of *assets* value
- Null values
  - Can use null values, but they are difficult to handle.

Redundancy creates problems

- Anomalies (by Codd)
  - *Insertion anomaly*: cannot store information about a branch if no loans exist
  - *Deletion anomaly*: lose branch info when that last account for the branch is deleted
  - *Update anomaly*: what happens when you modify asset for a branch in only a single record?
- The problems are caused by redundancy!
- Solution => decompose schema so that each information content is represented only once (later)
  - information content: relationship between attributes

First Normal Form

- Domain is atomic if its elements are considered to be indivisible units
  - Examples of non-atomic domains:
    - set of names, composite attributes
    - identification numbers like CS101 that can be broken up into parts
- A relational schema is in ***first normal form (1NF)*** if the domains of all attributes are atomic
- Atomicity is actually a property of how the elements of the domain are used
  - Student ID numbers: *CS0012, EE1127, …*
  - If the first two characters are extracted to find the department => the domain is not atomic
- Non-atomic attributes
  - leads to encoding of information in application program rather than in the database
  - complicate storage and query processing
- We assume all relations are in first normal form

Relational Theory

Goal: Devise a theory for the following

- Decide whether a particular relation R is in "good" form.
- In the case that a relation R is not in "good" form, decompose it into a set of relations $\{R_1, R_2, ..., R_n\}$ such that
  - each relation is in good form
  - the decomposition is lossless (preserves the information in the original relation before decomposition)
- Our theory is based on:
  - functional dependencies
  - multivalued dependencies (not covered in this semester)

Functional Dependencies

- Constraints on the set of legal relations.
- Require that the value for a certain set of attributes determines uniquely the value for another set of attributes.
- A functional dependency is a generalization of the notion of a *key*.

Functional Dependencies (Cont.)

- Let R be a relation schema

  $$\alpha \subseteq R \quad \textit{and} \quad \beta \subseteq R$$

- The *functional dependency* $\alpha \rightarrow \beta$ holds on R if and only if
  - for any *legal* relations $r(R)$,
  - whenever any two tuples t_1 and t_2 of r agree on the attributes α,
  - they also agree on the attributes β.
  - That is,

    $$t_1[\alpha] = t_2[\alpha] \quad \Rightarrow \quad t_1[\beta] = t_2[\beta]$$

- Example
  - Consider $r(A,B)$ with the following instance of r

1	4
1	5
3	7

  - On this instance, $A \rightarrow B$ does **NOT** hold, but $B \rightarrow A$ does hold

Applications of FD

- K is a superkey for relation schema R if and only if $K \rightarrow R$
- K is a candidate key for R if and only if
  - $K \rightarrow R$, and
  - for no $\alpha \subset K$, $\alpha \rightarrow R$
- Functional dependencies allow us to express constraints that cannot be expressed using superkeys.

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

  We expect the following functional dependencies to hold:

  loan-number $\rightarrow$ *amount*
  loan-number $\rightarrow$ *branch-name*

  but would not expect the following to hold:

  loan-number $\rightarrow$ *customer-name*

Applications of FD (Cont.)

- Specify constraints on the set of legal relations
  - We say that *F* holds on *R* if all legal relations on *R* satisfy the set of functional dependencies *F*.
- Test relations to see if they are legal under a given set of FDs
  - If a relation *r* is legal under a set *F* of functional dependencies, we say that *r* satisfies *F*.

- Note: A specific instance of a relation schema may satisfy a functional dependency even if the functional dependency does not hold on all legal instances.

Trivial FD

- *A* functional dependency is trivial if it is satisfied by all instances of a relation
  - *E.g.*
    - *customer-name, loan-number* $\rightarrow$ *customer-name*
    - *customer-name* $\rightarrow$ *customer-name*

- Lemma: $\alpha \rightarrow \beta$ is trivial if $\beta \subseteq \alpha$

Closure of a Set of FDs

- Given a set F of FDs, there are certain other FDs that are logically implied by F
  - E.g. If $A \rightarrow B$ and $B \rightarrow C$, then we can infer that $A \rightarrow C$
- The set of all functional dependencies logically implied by F is the *closure* of F.
- We denote the *closure* of F by F^+
- We can find all of F^+ by applying *Armstrong's Axioms*:
  - if $\beta \subseteq \alpha$, then $\alpha \rightarrow \beta$ **(*reflexivity*)**
  - if $\alpha \rightarrow \beta$, then $\gamma\ \alpha \rightarrow \gamma\ \beta$ **(*augmentation*)**
  - if $\alpha \rightarrow \beta$, and $\beta \rightarrow \gamma$, then $\alpha \rightarrow \gamma$ **(*transitivity*)**
- These rules are
  - sound (generate only functional dependencies that actually hold) and
  - complete (generate all functional dependencies that hold).

Example

- $R = (A, B, C, G, H, I)$
  $F = \{ A \rightarrow B$
  $A \rightarrow C$
  $CG \rightarrow H$
  $CG \rightarrow I$
  $B \rightarrow H\}$
- some members of F^+
  - $A \rightarrow H$
    - by transitivity from $A \rightarrow B$ *and* $B \rightarrow H$
  - $AG \rightarrow I$
    - by augmenting $A \rightarrow C$ with G, to get $AG \rightarrow CG$
      and then transitivity with $CG \rightarrow I$
  - $CG \rightarrow HI$
    - from $CG \rightarrow H$ *and* $CG \rightarrow I$: "union rule" can be inferred from
      - definition of functional dependencies, or
      - Augmentation of $CG \rightarrow I$ to infer $CG \rightarrow CGI$, augmentation of $CG \rightarrow H$ to infer $CGI \rightarrow HI$, and then transitivity

Decomposition

- Redundancy causes problems: Solution => decompose schema so that each information content is represented only once
- Definition: Let R be a relation scheme

  $\{R_1, ..., R_n\}$ is a decomposition of R if $R = R_1 \cup ... \cup R_n$

  (i.e., all of R's attributes are represented)
- We will deal mostly with binary decomposition:
  - R into $\{R_1, R_2\}$ where $R = R_1 \cup R_2$

student(*ID, name, dept, dept_chair, dept_phone, year*)

=> *student'*(*ID, name, year, dept*)

department(*dept, chair, phone*)

Lending = (*b_name, asset, b_city, loan#, c_name, amount*)

=> *Branch* = (*b_name, asset, b_city*)

Loan = (*loan#, c_name, amount*)

Lossy Decomposition

- Careless decomposition leads to loss of information: Lossy decomposition

Lending = (*b_name, asset, b_city, loan#, c_name, amount*)

=> *Branch* = (*b_name, asset, b_city*)

Loan = (*loan#, c_name, amount*)

- problem: relationship between *loan* and *branch* is lost
- loss of information

=> *Branch* = (*b_name, asset, b_city*)

Loan = (*loan#, c_name, amount, b_city*)

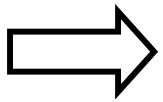
- more tuples in natural join
- but we have lost the relationship
- loss of information

Lossy Decomposition (cont.)

- Decomposition of $R = (A, B)$ into

  $R_1 = (A)$ and $R_2 = (B)$

A	B
α	1
α	2
β	1



A
α
β

$\Pi_A(r)$

B
1
2

$\Pi_B(r)$

- Can we *recover* the *original information content*?

$\Pi_A(r) \bowtie \Pi_B(r)$

A	B
α	1
α	2
β	1
β	2

Lossy!

Lossless-join Decomposition

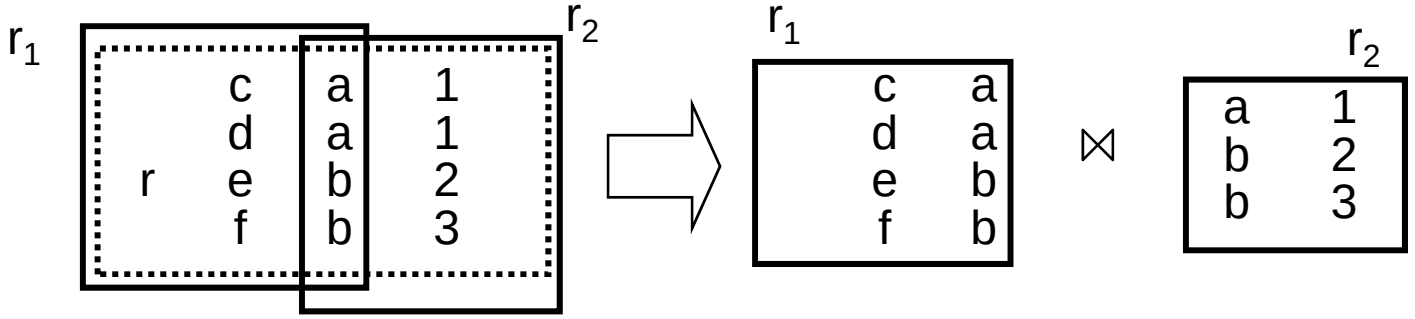
- For $r(R)$ and decomposition $\{R_1, R_2\}$, it is always the case that

$$r \subseteq \Pi_{R1}(r) \bowtie \Pi_{R2}(r)$$

- *Definition:* Decomposition $\{R_1, R_2\}$ is a *lossless-join decomposition* of R if

$$r = \Pi_{R1}(r) \bowtie \Pi_{R2}(r)$$

  - The information content of the original relation r is always the basis

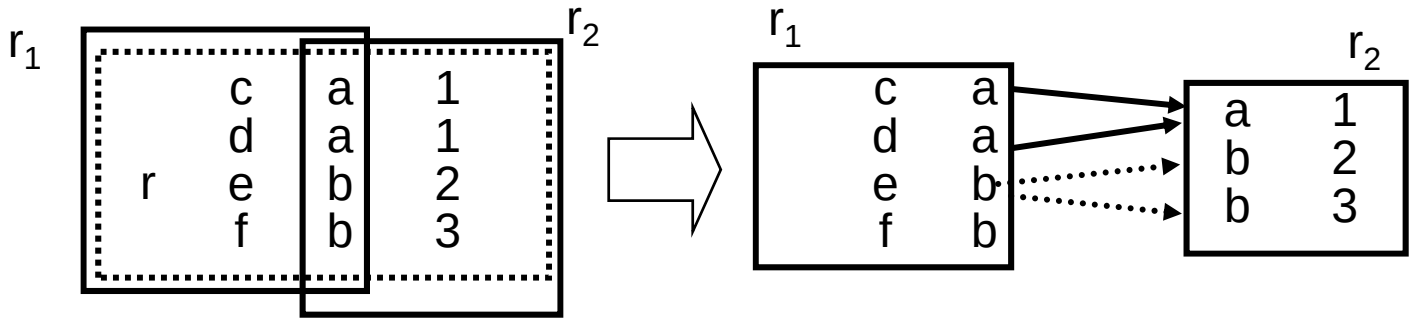


Lossless-join Decomposition

- *Lemma:* $\{R_1,...,R_n\}$ is a lossless decomposition if

$$R_1 \cap R_2 \rightarrow R_1, \quad \text{or} \quad R_1 \cap R_2 \rightarrow R_2$$

  - i.e., if one of the two subschemas hold the key of the other subschema



Boyce-Codd Normal Form

- We want a way to decide whether a particular relation R is in "good" form.
- Definition: A relation schema R is in BCNF (with respect to a set F of FDs) if for each FD $\alpha \rightarrow \beta$ in F^+ ($\alpha \subseteq R$ and $\beta \subseteq R$), at least one of the following holds:
  - $\alpha \rightarrow \beta$ is trivial (i.e., $\beta \subseteq \alpha$)
  - α is a superkey for R
- Example

  $R = (A, B, C), \quad F = \{A \rightarrow B ; \ B \rightarrow C\}, \quad \text{Key} = \{A\}$

  - R is not in BCNF
  - Decompose into $R_1 = (A, B), \ R_2 = (B, C)$
    - R_1 and R_2 in BCNF
    - Lossless-join decomposition

BCNF Example

- $R = (bname, bcity, assets, cname, loan\#, amount)$

  $F = \{ bname \rightarrow assets\ bcity ; \quad loan\# \rightarrow amount\ bname \}$

  Key = $\{loan\#, cname\}$

Decomposition

$R_1 = (bname, bcity, assets)$

$R_2 = (bname, cname, loan\#, amount)$

$R_3 = (bname, loan\#, amount)$

$R_4 = (cname, loan\#)$

Final decomposition result: $\{ R_1, R_3, R_4 \}$

Dependency Preservation

- Example *student*{*name, dept, college*}

  name → *dept*, *college*

  dept→ *college*

- Decomposition 1

  student1(*name, dept*) *name* → *dept*

  department(*dept, college*) *dept* → *college*

- Decomposition 2

  student1(*name, dept*) *name* → *dept*

  student2(*name, college*) *name* → *college*

  - is a lossless decomposition
  - but in order to test *dept*→ *college*, a join is required

Dependency Preservation (cont.)

- *Definition*

  F : set of FD on R. $\{R_1, ..., R_n\}$: decomposition of R.

  F_i : *restriction* of F to R_i is the set of all F.D.s in F^+ that include only attributes of R_i

- *Definition*

  Let $F' = F_1 \cup \ldots \cup F_n$. The decomposition is *dependency-preserving* if $F^+ = F'^+$

- *Motivation*: We wish to guarantee F by locally enforcing the each restriction (R_i) on the respective decomposed relation.
  - SQL does not provide a direct way of specifying functional dependencies other than superkeys.
  - Using assertions can be expensive

Example

- $R = (A, B, C)$
  $F = \{ A \rightarrow B,\ B \rightarrow C \}$
- $R_1 = (A, B),\quad R_2 = (B, C)$
  - Lossless-join decomposition: $R_1 \cap R_2 = \{B\}$ and $B \rightarrow BC$
  - Dependency preserving
- $R_1 = (A, B),\quad R_2 = (A, C)$
  - Lossless-join decomposition: $R_1 \cap R_2 = \{A\}$ and $A \rightarrow AB$
  - Not dependency preserving
    (cannot check $B \rightarrow C$ without computing $R_1 \bowtie R_2$)

BCNF and Dependency Preservation

- $R = (J, K, L)$
  $F = \{ JK \rightarrow L;\ L \rightarrow K \}$
  Two candidate keys = JK and JL
  - R is not in BCNF
  - Any decomposition of R will fail to preserve
    $JK \rightarrow L$
- *It is not always possible to get a BCNF decomposition that is dependency preserving*
- There are some situations where
  - BCNF is not dependency preserving, and
  - efficient checking for FD violation on updates is important

  => *solution:* define a weaker normal form

Third Normal Form

- Third Normal Form
  - Allows some redundancy (with resultant problems)
  - But FDs can be checked on individual relations without computing a join
  - There is always a lossless-join, dependency-preserving decomposition into 3NF

- A relation schema R is in *third normal form* (*3NF*) if

  for all $\alpha \rightarrow \beta$ in F^+ at least one of the following holds:
  - $\alpha \rightarrow \beta$ is trivial (i.e., $\beta \in \alpha$)
  - α is a superkey for R
  - Each attribute A in $\beta - \alpha$ is contained in a candidate key for R.

    (*NOTE:* each attribute may be in a different candidate key)

- If a relation is in BCNF it is in 3NF
  - since in BCNF one of the first two conditions above must hold

Example

$R = (J, K, L)$

$F = \{ JK \rightarrow L, L \rightarrow K \}$

- Two candidate keys: JK and JL
- R is in 3NF

  $JK \rightarrow L$ JK is a superkey

  $L \rightarrow K$ K is contained in a candidate key

- There is some redundancy in this schema
- BCNF decomposition has (JL) and (LK)

  => Testing for $JK \rightarrow L$ requires a join

R(*street, city, zip*)

street, city → *zip*

zip → *city*

St	*Zp*	*C*
s_1	z_1	c_1
s_2	z_1	c_1
s_3	z_2	c_1
null	z_3	c_2

/* 3NF but not in BCNF (nontrivial & *zip* is not key) */

- repetition of information (e.g., the relationship z_1, c_1)
- need to use null values (e.g., to represent the relationship z_3, c_2 where there is no corresponding value for *St*)

R1(*street, zip*)

R2(*zip, city*)

*R*1, *R*2 are in BCNF but not dependency-preserving.

St	*Zp*
s_1	z_1
s_2	z_1
s_3	z_2

Zp	*C*
z_1	c_1
z_2	c_1
z_3	c_2

Comparison of BCNF and 3NF

- It is always possible to decompose a relation into relations in 3NF and
  - the decomposition is lossless
  - the dependencies are preserved

- It is always possible to decompose a relation into relations in BCNF and
  - the decomposition is lossless
  - it *may not* be possible to preserve dependencies.

Design Goals

- When we decompose a relation schema R with a set of functional dependencies F into R_1, R_2,.., R_n we want
  1. Lossless decomposition
  2. No redundancy
  3. Dependency preservation

- First, try to achieve
  - BCNF
  - Lossless join
  - Dependency preservation

- If we cannot achieve this, we accept one of
  - Lack of dependency preservation
  - Redundancy due to use of 3NF

Algorithms

- Testing for BCNF
- BCNF Decomposition
- Testing for 3NF
- 3NF Decomposition

- Closure of FDs
- Closure of attributes
- Cover
- Canonical cover

Closure of Attribute Sets

- Given a set of attributes α, define the *closure* of α under F (denoted by α^+) as the set of attributes that are functionally determined by α under F:

$$\alpha \rightarrow \beta \text{ is in } F^+ \Leftrightarrow \beta \subseteq \alpha^+$$

- Algorithm to compute α^+, the closure of α under F

```
result := α;
while (changes to result) do
    for each β → γ in F do
      begin
        if β ⊆ result then result := result ∪ γ
      end
```

Example

- $R = (A, B, C, G, H, I)$
- $F = \{ A \rightarrow B;\ A \rightarrow C;\quad CG \rightarrow H;\quad CG \rightarrow I;\quad B \rightarrow H \}$

- $(AG)^+$
  1. *result* = AG
  2. *result* = $ABCG$ $(A \rightarrow C$ and $A \rightarrow B)$
  3. *result* = $ABCGH$ $(CG \rightarrow H$ and $CG \subseteq AGBC)$
  4. *result* = $ABCGHI$ $(CG \rightarrow I$ and $CG \subseteq AGBCH)$

- Is AG a candidate key?
  - Is AG a super key?
    - Does $AG \rightarrow R$?
  - Is any subset of AG a superkey?
    - Does $A^+ \rightarrow R$?
    - Does $G^+ \rightarrow R$?

Uses of Attribute Closure

There are several uses of the attribute closure algorithm:

- Testing for superkey:
  - To test if α is a superkey, we compute α^+, and check if α^+ contains all attributes of R.
- Testing functional dependencies
  - To check if a functional dependency $\alpha \rightarrow \beta$ holds (or, in other words, is in F^+), just check if $\beta \subseteq \alpha^+$.
  - Is a very useful simple test
- Computing closure of F
  - For each $\gamma \subseteq R$, we find the closure γ^+, and
  - for each $S \subseteq \gamma^+$, we output a functional dependency $\gamma \rightarrow S$.

Testing for BCNF

- Check if $\alpha \rightarrow \beta$ cause a violation of BCNF
  1. compute α^+ (the attribute closure of α), and
  2. verify that it includes all attributes of R (i.e., it is a superkey of R)

- Check if R is in BCNF (w.r.t. F)

  Simplified test: check only the dependencies in F for violation of BCNF, rather than checking all dependencies in F^+
  - It can be shown that if none of the dependencies in F causes a violation of BCNF, then none of the dependencies in F^+ will cause a violation of BCNF either

Testing for BCNF (cont.)

- However, using only F is incorrect when testing a relation in a decomposition of R

- Example

  Consider R (A, B, C, D)

  with $F = \{ A \rightarrow B,\ B \rightarrow C \}$

  - Decompose into R_1(A, B) and R_2(A, C, D)
  - Neither of the dependencies in F contain only attributes from (A, C, D) so we might be mislead into thinking R_2 satisfies BCNF.
  - In fact, dependency $A \rightarrow C$ in F^+ shows R_2 is not in BCNF.

Cover

- Sets of functional dependencies may have redundant dependencies that can be inferred from the others
  - $A \rightarrow C$ is redundant in: $\{ A \rightarrow B, \quad B \rightarrow C, \quad A \rightarrow C \}$
  - Parts of a functional dependency may be redundant
    - E.g. on RHS: $\{ A \rightarrow B, \quad B \rightarrow C, \quad A \rightarrow CD \}$ can be simplified to $\{ A \rightarrow B, \quad B \rightarrow C, \quad A \rightarrow D \}$
    - E.g. on LHS: $\{ A \rightarrow B, \quad B \rightarrow C, \quad AC \rightarrow D \}$ can be simplified to $\{ A \rightarrow B, \quad B \rightarrow C, \quad A \rightarrow D \}$

- A <u>cover</u> of F is any F' such that $F'^{+} = F^{+}$
- A FD $g \in F$ is <u>redundant</u> if

  $(F - \{g\})^{+} = F+$ or $g \in (F - \{g\})^{+}$
- F' is a <u>nonredundant (minimal) cover</u> of F if
  - $F'^{+} = F^{+}$ and
  - F' contains no redundant FD

Extraneous Attributes

- Let $\alpha \rightarrow \beta$ in F.
  - $A \in \alpha$ is extraneous if $F \Rightarrow (F - \{\alpha \rightarrow \beta\}) \cup \{(\alpha - A) \rightarrow \beta\}$
  - $A \in \beta$ is extraneous if $F \Leftarrow (F - \{\alpha \rightarrow \beta\}) \cup \{\alpha \rightarrow (\beta - A)\}$
  - *Note:* implication in the opposite direction is trivial in each of the cases above, since a "stronger" functional dependency always implies a weaker one
- Example
  - Given $F = \{A \rightarrow C, AB \rightarrow C\}$
  - B is extraneous in $AB \rightarrow C$ because
    - $A \rightarrow C$ logically implies $AB \rightarrow C$
- Example
  - Given $F = \{A \rightarrow C, AB \rightarrow CD\}$
  - C is extraneous in $AB \rightarrow CD$ since
    - $A \rightarrow C$ can be inferred even after deleting C

Testing if an Attribute is Extraneous

- Consider a set F of functional dependencies and the functional dependency $\alpha \rightarrow \beta$ in F.
- To test if attribute $A \in \alpha$ is extraneous in α
  1. compute $(\{\alpha\} - A)^+$ using the dependencies in F
  2. check that $(\{\alpha\} - A)^+$ contains A; if it does, A is extraneous
- To test if attribute $A \in \beta$ is extraneous in β
  1. compute α^+ using only the dependencies in
     $$F' = (F - \{\alpha \rightarrow \beta\}) \cup \{\alpha \rightarrow (\beta - A)\},$$
  2. if α^+ contains A, A is extraneous

Canonical Cover

- A *canonical cover* for F is a set of dependencies F_c such that
  - $F_c^+ = F^+$
  - No FD in F_c contains an extraneous attribute
  - Each left side of a FD in F_c is unique
- Intuitively, a canonical cover of F is a "minimal" set of functional dependencies equivalent to F, with no redundant dependencies or having redundant parts of dependencies
- To *compute a canonical cover for* F:

  repeat

  Use the *union rule* to replace any dependencies in F

  $\alpha_1 \rightarrow \beta_1$ and $\alpha_1 \rightarrow \beta_1$ with $\alpha_1 \rightarrow \beta_1 \beta_2$

  Find a functional dependency $\alpha \rightarrow \beta$ with an

  extraneous attribute either in α or in β

  delete the extraneous attribute from $\alpha \rightarrow \beta$

  until F does not change

  - Union rule may become applicable after some extraneous attributes have been deleted, so it has to be re-applied
  - $O(n^2)$

Example

- $R = (A, B, C)$
  $F = \{ A \rightarrow BC$
  $B \rightarrow C$
  $A \rightarrow B$
  $AB \rightarrow C \}$
- Combine $A \rightarrow BC$ and $A \rightarrow B$ into $A \rightarrow BC$
  - Set is now $\{ A \rightarrow BC, B \rightarrow C, AB \rightarrow C \}$
- A is extraneous in $AB \rightarrow C$ because $B \rightarrow C$ logically implies $AB \rightarrow C$.
  - Set is now $\{ A \rightarrow BC, B \rightarrow C \}$
- C is extraneous in $A \rightarrow BC$ since $A \rightarrow BC$ is logically implied by $A \rightarrow B$ and $B \rightarrow$ C.
- The canonical cover is:

  $A \rightarrow B$
  $B \rightarrow C$

BCNF Decomposition Algorithm

result := {*R*}; *done* := false; compute F^+;
while (not *done)* **do**
if (there is a schema R_i in *result* that is not in BCNF)
then begin
let $\alpha \rightarrow \beta$ ($\alpha \cap \beta = \varnothing$) be a nontrivial FD
that holds on R_i, and $\alpha \rightarrow R_i$ is not in F^+,
result := (*result* – R_i) ∪ (R_i – β) ∪ (α, β);
end
else *done* := **true;**

- Note: each R_i in *result* is in BCNF, and decomposition is lossless-join.

- *R* = (*bname, bcity, assets, cname, loan#, amount*)
  F = { *bname* → *assets bcity* ; *loan#* → *amount bname* } Key = {*loan#, cname*}

  Decomposition
  - R_1 = (*bname, bcity, assets*), R_2 = (*bname, cname, loan#, amount*)
  - R_3 = (*bname, loan#, amount*), R_4 = (*cname, loan#*)

  Final decomposition result: R_1, R_3, R_4

Testing for 3NF

- Need to check only FDs in F (not $F^{+)}$
- Use attribute closure to check, for each dependency $\alpha \rightarrow \beta$, if α is a superkey.
- If α is not a superkey, we have to verify if each attribute in β is contained in a candidate key of R
  - this test is rather more expensive, since it involves finding candidate keys
  - testing for 3NF has been shown to be NP-hard
  - Interestingly, decomposition into third normal form can be done in polynomial time

3NF Decomposition Algorithm

Let F_c be a canonical cover for F;
$i := 0$;
for each FD $\alpha \rightarrow \beta$ in F_c **do**
  if none of R_j, $1 \le j \le i$ contains $\alpha\ \beta$ **then**
    { $i := i + 1$
      $R_i := \alpha\ \beta$ }
if none of R_j, $1 \le j \le i$ contains a candidate key for R
  then { $i := i + 1$;
    R_i := any candidate key for R }
return (R_1, R_2, ..., R_i)

- *Example:* *Banker* = (*branch, cname, banker, office#*)
  FD: { *banker* → *branch office#* ; *cname branch* → *banker* }
  The key: {*cname, branch*}

  Follow the algorithm

  Banker1 = (*banker, branch, office#*)

  Banker2 = (*cname, branch, banker*)

  Since *Banker2* contains a candidate key we are done.

Overall Database Design Process

- We have assumed schema *R* is given
- *R* could have been a single relation containing *all* attributes that are of interest (called ***universal relation***).
  - Normalization breaks *R* into smaller relations.
- *R* could have been generated when converting E-R diagram to a set of tables.
- *R* could have been the result of some ad hoc design of relations, which we then test/convert to normal form.

ER Model and Normalization

- When an E-R diagram is carefully designed, identifying all entities correctly, the tables generated from the E-R diagram should not need further normalization.
- However, in a real (imperfect) design there can be FDs from non-key attributes of an entity to other attributes of the entity
  - E.g. *employee* entity with attributes *department-number* and *department-address*, and an FD *department-number* $\rightarrow$ *department-address*
  - Good design would have made *department* an entity

Denormalization for Performance

- May want to use non-normalized schema for performance
  - E.g. displaying *customer-name* along with *account-number* and *balance* requires join of *account* with *depositor*
- Alternative 1: Use denormalized relation containing attributes of *account* as well as *depositor* with all above attributes
  - faster lookup
  - Extra space and extra execution time for updates
  - extra coding work for programmer and possibility of error in extra code
- Alternative 2: use a *materialized view* defined as

  $$account \bowtie depositor$$

  - Benefits and drawbacks same as above, except no extra coding work for programmer and avoids possible errors

Other Design Issues

- Some aspects of database design are not caught by normalization
- Examples of bad database design, to be avoided:

  Instead of *earnings*(*company-id, year, amount*), use
  - *earnings-2000, earnings-2001, earnings-2002*, etc., all on the schema (*company-id, earnings*).
    - Above are in BCNF, but make querying across years difficult and needs new table each year
  - *company-year*(*company-id, earnings-2000, earnings-2001, earnings-2002*)
    - Also in BCNF, but also makes querying across years difficult and requires new attribute each year.
    - Is an example of a ***crosstab***, where values for one attribute become column names
    - Used in spreadsheets, and in data analysis tools

END OF CHAPTER 7