



Lecture 17 of 42

$r(A) = s(S)$

More Normal Forms

Notes: 3NF vs. BCNF vs. 4NF; MP4, Exam 1



Wednesday, 04 October 2006

William H. Hsu

Department of Computing and Information Sciences, KSU

KSOL course page: <http://snipurl.com/va60>

Course web site: <http://www.kddresearch.org/Courses/Fall-2006/CIS560>

Instructor home page: <http://www.cis.ksu.edu/~bhsu>

SER

Reading for Next Class:

Second half of Chapter 7, Silberschatz *et al.*, 5th edition



Chapter 7: Relational Database Design

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



Functional Dependencies: Review

- Let R be a relation schema
 $\alpha \subseteq R$ and $\beta \subseteq R$
- 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,
 $t_1[\alpha] = t_2[\alpha] \Rightarrow t_1[\beta] = t_2[\beta]$
- 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.



Functional Dependencies Example: Review

- 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. Consider the schema:

$bor_loan = (\underline{customer_id}, \underline{loan_number}, amount)$.

We expect this functional dependency to hold:

$loan_number \rightarrow amount$

but would not expect the following to hold:

$amount \rightarrow customer_name$





Boyce-Codd Normal Form: Review

A relation schema R is in BCNF with respect to a set F of functional dependencies if for all functional dependencies in F^+ of the form

$$\alpha \rightarrow \beta$$

where $\alpha \subseteq R$ 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 schema *not* in BCNF:

$bor_loan = (customer_id, loan_number, amount)$

because $loan_number \rightarrow amount$ holds on bor_loan but $loan_number$ is not a superkey



Decomposing a Schema into BCNF: Review

- Suppose we have a schema R and a non-trivial dependency $\alpha \rightarrow \beta$ causes a violation of BCNF.

We decompose R into:

- $(\alpha \cup \beta)$
- $(R - (\beta - \alpha))$
- In our example,
 - * $\alpha = loan_number$
 - * $\beta = amount$and bor_loan is replaced by
 - * $(\alpha \cup \beta) = (loan_number, amount)$
 - * $(R - (\beta - \alpha)) = (customer_id, loan_number)$





BCNF and Dependency Preservation

- Constraints, including functional dependencies, are costly to check in practice unless they pertain to only one relation
- If it is sufficient to test only those dependencies on each individual relation of a decomposition in order to ensure that *all* functional dependencies hold, then that decomposition is *dependency preserving*.
- Because it is not always possible to achieve both BCNF and dependency preservation, we consider a weaker normal form, known as *third normal form*.



Third Normal Form

- 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).
- Third condition is a minimal relaxation of BCNF to ensure dependency preservation (will see why later).





Goals of Normalization

- Let R be a relation scheme with a set F of functional dependencies.
- Decide whether a relation scheme R is in “good” form.
- In the case that a relation scheme R is not in “good” form, decompose it into a set of relation scheme $\{R_1, R_2, \dots, R_n\}$ such that
 - * each relation scheme is in good form
 - * the decomposition is a lossless-join decomposition
 - * Preferably, the decomposition should be dependency preserving.



How good is BCNF?

- There are database schemas in BCNF that do not seem to be sufficiently normalized
- Consider a database
classes (course, teacher, book)

such that $(c, t, b) \in \text{classes}$ means that t is qualified to teach c , and b is a required textbook for c
- The database is supposed to list for each course the set of teachers any one of which can be the course’s instructor, and the set of books, all of which are required for the course (no matter who teaches it).





How good is BCNF? (Cont.)

<i>course</i>	<i>teacher</i>	<i>book</i>
database	Avi	DB Concepts
database	Avi	Ullman
database	Hank	DB Concepts
database	Hank	Ullman
database	Sudarshan	DB Concepts
database	Sudarshan	Ullman
operating systems	Avi	OS Concepts
operating systems	Avi	Stallings
operating systems	Pete	OS Concepts
operating systems	Pete	Stallings

classes

- There are no non-trivial functional dependencies and therefore the relation is in BCNF
- Insertion anomalies – i.e., if Marilyn is a new teacher that can teach database, two tuples need to be inserted

(database, Marilyn, DB Concepts)
(database, Marilyn, Ullman)



How good is BCNF? (Cont.)

- Therefore, it is better to decompose *classes* into:

<i>course</i>	<i>teacher</i>
database	Avi
database	Hank
database	Sudarshan
operating systems	Avi
operating systems	Jim

teaches

<i>course</i>	<i>book</i>
database	DB Concepts
database	Ullman
operating systems	OS Concepts
operating systems	Shaw

text

This suggests the need for higher normal forms, such as Fourth Normal Form (4NF), which we shall see later.





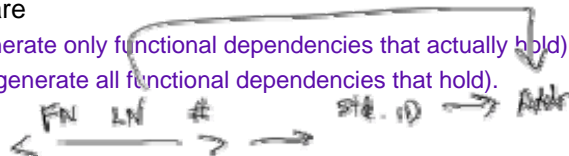
Functional-Dependency Theory

- We now consider the formal theory that tells us which functional dependencies are implied logically by a given set of functional dependencies.
- We then develop algorithms to generate lossless decompositions into BCNF and 3NF
- We then develop algorithms to test if a decomposition is dependency-preserving



Closure of a Set of Functional Dependencies

- Given a set F set of functional dependencies, there are certain other functional dependencies that are logically implied by F .
 - * For example: 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^+
 - * $A \rightarrow H$
 - \Rightarrow by transitivity from $A \rightarrow B$ and $B \rightarrow H$
 - * $AG \rightarrow I$
 - \Rightarrow by augmenting $A \rightarrow C$ with G , to get $AG \rightarrow CG$
 - and then transitivity with $CG \rightarrow I$
 - * $CG \rightarrow HI$
 - \Rightarrow by augmenting $CG \rightarrow I$ to infer $CG \rightarrow CGI$,
 - and augmenting of $CG \rightarrow H$ to infer $CGI \rightarrow HI$,
 - and then transitivity



Procedure for Computing F^+

- To compute the closure of a set of functional dependencies F :

$F^+ = F$

repeat

- for each** functional dependency f in F^+
 - apply reflexivity and augmentation rules on f
 - add the resulting functional dependencies to F^+
 - for each** pair of functional dependencies f_1 and f_2 in F^+
 - if** f_1 and f_2 can be combined using transitivity
 - then** add the resulting functional dependency to F^+
- until** F^+ does not change any further

NOTE: We shall see an alternative procedure for this task later





Closure of Functional Dependencies (Cont.)

- We can further simplify manual computation of F^+ by using the following additional rules.
 - * If $\alpha \rightarrow \beta$ holds and $\alpha \rightarrow \gamma$ holds, then $\alpha \rightarrow \beta\gamma$ holds (**union**)
 - * If $\alpha \rightarrow \beta\gamma$ holds, then $\alpha \rightarrow \beta$ holds and $\alpha \rightarrow \gamma$ holds (**decomposition**)
 - * If $\alpha \rightarrow \beta$ holds and $\gamma\beta \rightarrow \delta$ holds, then $\alpha\gamma \rightarrow \delta$ holds (**pseudotransitivity**)

The above rules can be inferred from Armstrong's axioms.



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

```
result :=  $\alpha$ ;  
while (changes to result) do  
  for each  $\beta \rightarrow \gamma$  in  $F$  do  
    begin  
      if  $\beta \subseteq \text{result}$  then  $\text{result} := \text{result} \cup \gamma$   
    end
```





Example of Attribute Set Closure

- $R = (A, B, C, G, H, I)$
- $F = \{A \rightarrow B, A \rightarrow C, CG \rightarrow H, CG \rightarrow I, B \rightarrow H\}$
- $(AG)^+$
 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?
 1. Is AG a super key?
 1. Does $AG \rightarrow R? \iff \text{Is } (AG)^+ \supseteq R$
 2. Is any subset of AG a superkey?
 1. Does $A \rightarrow R? \iff \text{Is } (A)^+ \supseteq R$
 2. Does $G \rightarrow R? \iff \text{Is } (G)^+ \supseteq R$



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 $\beta \subseteq \alpha^+$.
 - * That is, we compute α^+ by using attribute closure, and then check if it contains β .
 - * Is a simple and cheap test, and very useful
- 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 $\gamma \rightarrow S$.





Canonical Cover

- Sets of functional dependencies may have redundant dependencies that can be inferred from the others
 - * For example: $A \rightarrow C$ is redundant in: $\{A \rightarrow B, B \rightarrow C\}$
 - * Parts of a functional dependency may be redundant
 - ⇒ E.g.: on RHS: $\{A \rightarrow B, B \rightarrow C, A \rightarrow CD\}$ can be simplified to $\{A \rightarrow B, B \rightarrow C, A \rightarrow D\}$
 - ⇒ E.g.: on LHS: $\{A \rightarrow B, B \rightarrow C, AC \rightarrow D\}$ can be simplified to $\{A \rightarrow B, B \rightarrow C, A \rightarrow D\}$
- Intuitively, a canonical cover of F is a “minimal” set of functional dependencies equivalent to F , having no redundant dependencies or redundant parts of dependencies



3NF Example

- Relation R :
 - * $R = (J, K, L)$
 - * $F = \{JK \rightarrow L, L \rightarrow K\}$
 - * 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





Redundancy in 3NF

- There is some redundancy in this schema
- Example of problems due to redundancy in 3NF

* $R = (J, K, L)$
 $F = \{JK \rightarrow L, L \rightarrow K\}$

J	L	K
j_1	l_1	k_1
j_2	l_1	k_1
j_3	l_1	k_1
null	l_2	k_2

- repetition of information (e.g., the relationship l_1, k_1)
- need to use null values (e.g., to represent the relationship l_2, k_2 where there is no corresponding value for J).



Testing for 3NF

- Optimization: Need to check only FDs in F , need not check all FDs in 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 involve finding candidate keys
 - * testing for 3NF has been shown to be NP-hard
 - * Interestingly, decomposition into third normal form (described shortly) can be done in polynomial time





3NF Decomposition Algorithm

```
Let  $F_C$  be a canonical cover for  $F$ ;  
 $i := 0$ ;  
for each functional dependency  $\alpha \rightarrow \beta$  in  $F_C$  do  
  if none of the schemas  $R_j$ ,  $1 \leq j \leq i$  contains  $\alpha \beta$   
    then begin  
       $i := i + 1$ ;  
       $R_i := \alpha \beta$   
    end  
if none of the schemas  $R_j$ ,  $1 \leq j \leq i$  contains a candidate key for  $R$   
  then begin  
     $i := i + 1$ ;  
     $R_i :=$  any candidate key for  $R$ ;  
  end  
return  $(R_1, R_2, \dots, R_i)$ 
```



3NF Decomposition Algorithm (Cont.)

- Above algorithm ensures:
 - * each relation schema R_i is in 3NF
 - * decomposition is dependency preserving and lossless-join
 - * Proof of correctness is at end of this file ([click here](#))





Example

- Relation schema:
 $cust_banker_branch = (customer_id, employee_id, branch_name, type)$
- The functional dependencies for this relation schema are:
 $customer_id, employee_id \rightarrow branch_name, type$
 $employee_id \rightarrow branch_name$
- The **for** loop generates:
 $(customer_id, employee_id, branch_name, type)$
It then generates
 $(employee_id, branch_name)$
but does not include it in the decomposition because it is a subset of the first schema.



Comparison of BCNF and 3NF

- It is always possible to decompose a relation into a set of relations that are in 3NF such that:
 - * the decomposition is lossless
 - * the dependencies are preserved
- It is always possible to decompose a relation into a set of relations that are in BCNF such that:
 - * the decomposition is lossless
 - * it may not be possible to preserve dependencies.





Design Goals

- Goal for a relational database design is:
 - * BCNF.
 - * Lossless join.
 - * Dependency preservation.
- If we cannot achieve this, we accept one of
 - * Lack of dependency preservation
 - * Redundancy due to use of 3NF
- Interestingly, SQL does not provide a direct way of specifying functional dependencies other than superkeys.
Can specify FDs using assertions, but they are expensive to test
- Even if we had a dependency preserving decomposition, using SQL we would not be able to efficiently test a functional dependency whose left hand side is not a key.



Multivalued Dependencies (MVDs)

- Let R be a relation schema and let $\alpha \subseteq R$ and $\beta \subseteq R$.
The *multivalued dependency*

$$\alpha \twoheadrightarrow \beta$$

holds on R if in any legal relation $r(R)$, for all pairs for tuples t_1 and t_2 in r such that $t_1[\alpha] = t_2[\alpha]$, there exist tuples t_3 and t_4 in r such that:

$$\begin{aligned}t_1[\alpha] &= t_2[\alpha] = t_3[\alpha] = t_4[\alpha] \\t_3[\beta] &= t_1[\beta] \\t_3[R - \beta] &= t_2[R - \beta] \\t_4[\beta] &= t_2[\beta] \\t_4[R - \beta] &= t_1[R - \beta]\end{aligned}$$





MVD (Cont.)

- Tabular representation of $\alpha \twoheadrightarrow \beta$

	α	β	$R - \alpha - \beta$
t_1	$a_1 \dots a_i$	$a_{i+1} \dots a_j$	$a_{j+1} \dots a_n$
t_2	$a_1 \dots a_i$	$b_{i+1} \dots b_j$	$b_{j+1} \dots b_n$
t_3	$a_1 \dots a_i$	$a_{i+1} \dots a_j$	$b_{j+1} \dots b_n$
t_4	$a_1 \dots a_i$	$b_{i+1} \dots b_j$	$a_{j+1} \dots a_n$



Example

- Let R be a relation schema with a set of attributes that are partitioned into 3 nonempty subsets.

Y, Z, W

- We say that $Y \twoheadrightarrow Z$ (Y multidetermines Z) if and only if for all possible relations $r(R)$

then $\langle y_1, z_1, w_1 \rangle \in r$ and $\langle y_2, z_2, w_2 \rangle \in r$
Handwritten: Same Dep FR *Handwritten: Same Dep FR*

$\langle y_1, z_1, w_2 \rangle \in r$ and $\langle y_2, z_2, w_1 \rangle \in r$

- Note that since the behavior of Z and W are identical it follows that

$Y \twoheadrightarrow Z$ if $Y \twoheadrightarrow W$



Example (Cont.)

- In our example:

$course \twoheadrightarrow teacher$
 $course \twoheadrightarrow book$

- The above formal definition is supposed to formalize the notion that given a particular value of Y (*course*) it has associated with it a set of values of Z (*teacher*) and a set of values of W (*book*), and these two sets are in some sense independent of each other.
- Note:
 - * If $Y \rightarrow Z$ then $Y \twoheadrightarrow Z$
 - * Indeed we have (in above notation) $Z_1 = Z_2$
The claim follows.



Use of Multivalued Dependencies

- We use multivalued dependencies in two ways:
 1. To test relations to determine whether they are legal under a given set of functional and multivalued dependencies
 2. To specify constraints on the set of legal relations. We shall thus concern ourselves *only* with relations that satisfy a given set of functional and multivalued dependencies.
- If a relation r fails to satisfy a given multivalued dependency, we can construct a relations r' that does satisfy the multivalued dependency by adding tuples to r .





Theory of MVDs

- From the definition of multivalued dependency, we can derive the following rule:

- * If $\alpha \twoheadrightarrow \beta$, then $\alpha \rightarrow \beta$

That is, every functional dependency is also a multivalued dependency

- The **closure** D^+ of D is the set of all functional and multivalued dependencies logically implied by D .

- * We can compute D^+ from D , using the formal definitions of functional dependencies and multivalued dependencies.
- * We can manage with such reasoning for very simple multivalued dependencies, which seem to be most common in practice
- * For complex dependencies, it is better to reason about sets of dependencies using a system of inference rules (see Appendix C).

C T B



Don't want any α, β
 st. $\alpha \twoheadrightarrow \beta$ but
 $\neg (\alpha \rightarrow \beta)$



Fourth Normal Form

- A relation schema R is in 4NF with respect to a set D of functional and multivalued dependencies if for all multivalued dependencies in D^+ of the form $\alpha \twoheadrightarrow \beta$, where $\alpha \subseteq R$ and $\beta \subseteq R$, at least one of the following hold:

- * $\alpha \twoheadrightarrow \beta$ is trivial (i.e., $\beta \subseteq \alpha$ or $\alpha \cup \beta = R$)
- * α is a superkey for schema R

- If a relation is in 4NF it is in BCNF



Restriction of Multivalued Dependencies

- The restriction of D to R_i is the set D_i consisting of
 - * All functional dependencies in D^+ that include only attributes of R_i
 - * All multivalued dependencies of the form
$$\alpha \twoheadrightarrow (\beta \cap R_i)$$
where $\alpha \subseteq R_i$ and $\alpha \twoheadrightarrow \beta$ is in D^+



4NF Decomposition Algorithm

result := { R };

done := false;

compute D^+ ;

Let D_i denote the restriction of D^+ to R_i

while (not *done*)

if (there is a schema R_i in *result* that is not in 4NF) **then**

begin

 let $\alpha \twoheadrightarrow \beta$ be a nontrivial multivalued dependency that holds
 on R_i such that $\alpha \rightarrow R_i$ is not in D_i , and $\alpha \cap \beta = \emptyset$;

result := (*result* - R_i) \cup (R_i - β) \cup (α , β);

end

else *done* := true;

Note: each R_i is in 4NF, and decomposition is lossless-join





Example

- $R = (A, B, C, G, H, I)$
- $F = \{ A \twoheadrightarrow B$
 $B \twoheadrightarrow HI$
 $CG \twoheadrightarrow H \}$
- R is not in 4NF since $A \twoheadrightarrow B$ and A is not a superkey for R
- Decomposition
 - a) $R_1 = (A, B)$ (R_1 is in 4NF)
 - b) $R_2 = (A, C, G, H, I)$ (R_2 is not in 4NF)
 - c) $R_3 = (C, G, H)$ (R_3 is in 4NF)
 - d) $R_4 = (A, C, G, I)$ (R_4 is not in 4NF)
- Since $A \twoheadrightarrow B$ and $B \twoheadrightarrow HI$, $A \twoheadrightarrow HI$, $A \twoheadrightarrow I$
- e) $R_5 = (A, I)$ (R_5 is in 4NF)
- f) $R_6 = (A, C, G)$ (R_6 is in 4NF)



Further Normal Forms

- **Join dependencies** generalize multivalued dependencies
 - * lead to **project-join normal form (PJNF)** (also called **fifth normal form**)
- A class of even more general constraints, leads to a normal form called **domain-key normal form**.
- Problem with these generalized constraints: are hard to reason with, and no set of sound and complete set of inference rules exists.
- Hence rarely used

