

# LECTURE 16 OF 42

## Database Normalization Notes: Rationale for Normalization

SELECT  $A_1, A_2, \dots, A_k$   
FROM  $r_1, r_2, \dots, r_n$

Wednesday, 27 February 2008

WHERE  $P_1$  AND  $P_2$  AND...  $P_m$

William H. Hsu

Department of Computing and Information Sciences, KSU

$\equiv \pi_{A_1, A_2, \dots, A_k}(\sigma_{P_1 \wedge P_2 \wedge \dots \wedge P_m}(r_1 \times r_2 \times \dots \times r_n))$

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

Course web site: <http://www.kddresearch.org/Courses/Spring-2008/CIS560>

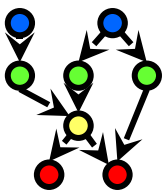
Instructor home page: <http://www.cis.ksu.edu/~bhsu>

$\bowtie \supseteq$   
 $\supseteq$   
 $\supseteq$   
 $\supseteq$   
 $\supseteq$   
 $\supseteq$   
 $\supseteq$   
 $\supseteq$   
 $\supseteq$   
 $\supseteq$

Reading for Next Class:

First half of Chapter 7, Silberschatz *et al.*, 5<sup>th</sup> 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

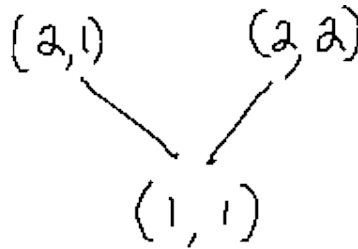
$$(a,b) < (c,d) \\ \Leftrightarrow a < b \\ \wedge c < d$$

student:

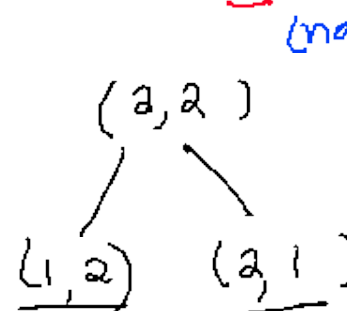
(wid, fn, ln)

Student:

(fn, ln, num, ...)



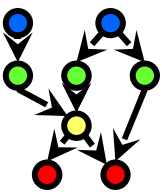
minimum



minimal  
(but not minimum)

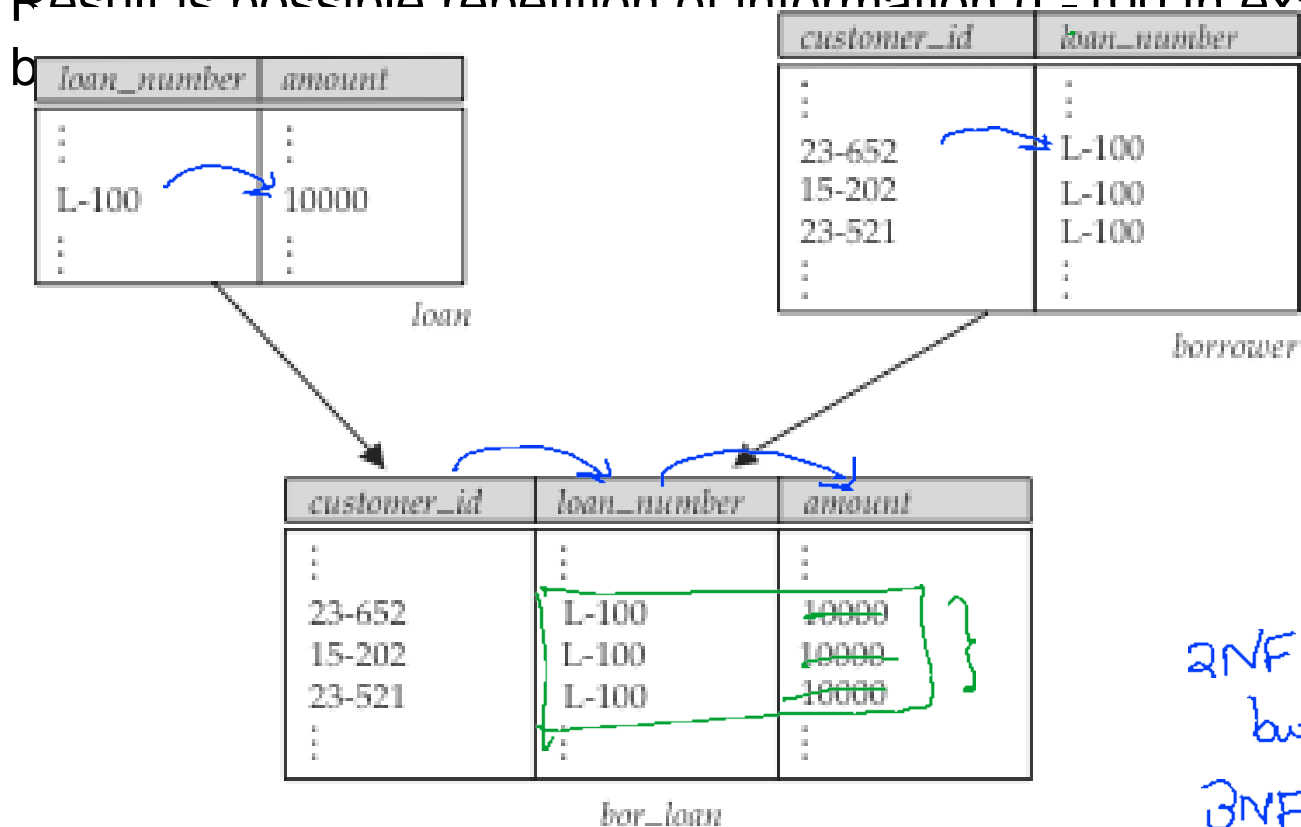
candidate key:  
minimal superkey  
= no smaller key

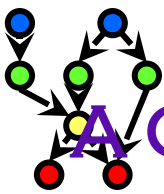




# COMBINE SCHEMAS?

- Suppose we combine *borrow* and *loan* to get  
*bor\_loan* = (*customer\_id*, *loan\_number*, *amount*)
- Result is possible repetition of information (L-100 in example)





# A COMBINED SCHEMA WITHOUT REPETITION

- Consider combining *loan\_branch* and *loan*  
*loan\_amt\_br* = (*loan\_number*, *amount*, *branch\_name*)
- No repetition (as suggested by example below)

<i>loan_number</i>	<i>amount</i>
⋮	⋮
L-100	10000
⋮	⋮

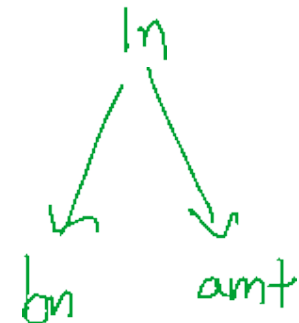
<i>loan_number</i>	<i>branch_name</i>
⋮	⋮
L-100	Springfield
⋮	⋮

*loan*

*loan\_branch*

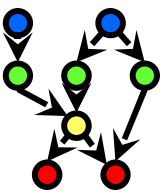
<i>loan_number</i>	<i>amount</i>	<i>branch_name</i>
⋮	⋮	⋮
L-100	10000	Springfield
⋮	⋮	⋮

*loan\_amt\_br*



3NF ✓

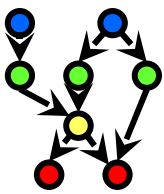




## WHAT ABOUT SMALLER SCHEMAS?

- Suppose we had started with *bor\_loan*. How would we know to split up (**decompose**) it into *borrower* and *loan*?
- Write a rule “if there were a schema (*loan\_number*, *amount*), then *loan\_number* would be a candidate key”
- Denote as a **functional dependency**:  
$$\textit{loan\_number} \rightarrow \textit{amount}$$
- In *bor\_loan*, because *loan\_number* is not a candidate key, the amount of a loan may have to be repeated. This indicates the need to decompose *bor\_loan*.
- Not all decompositions are good. Suppose we decompose *employee* into  
*employee1* = (*employee\_id*, *employee\_name*)  
*employee2* = (*employee\_name*, *telephone\_number*, *start\_date*)
- The next slide shows how we lose information -- we cannot reconstruct the original *employee* relation -- and so, this is a lossy decomposition.





# A LOSSY DECOMPOSITION

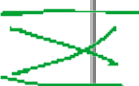
*A1*  
*B2*

<i>employee_id</i>	<i>employee_name</i>	<i>telephone_number</i>	<i>start_date</i>
⋮			
123-45-6789	Kim	882-0000	1984-03-29
987-65-4321	Kim	869-9999	1981-01-16
⋮			

*employee*

*A*  
*B*

<i>employee_id</i>	<i>employee_name</i>
⋮	
123-45-6789	Kim
987-65-4321	Kim
⋮	



<i>employee_name</i>	<i>telephone_number</i>	<i>start_date</i>
⋮		
<u>Kim</u>	882-0000	1984-03-29
<u>Kim</u>	869-9999	1981-01-16
⋮		

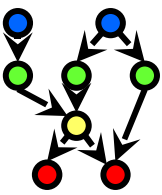
*1*  
*2*



*A1*  
~~*A1*~~  
~~*B1*~~  
*B2*

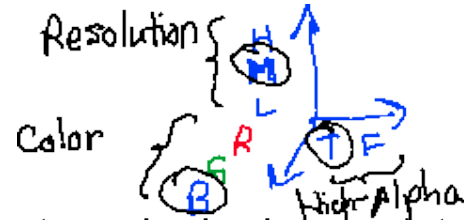
<i>employee_id</i>	<i>employee_name</i>	<i>telephone_number</i>	<i>start_date</i>
⋮			
123-45-6789	Kim	882-0000	1984-03-29
123-45-6789	Kim	869-9999	1981-01-16
987-65-4321	Kim	882-0000	1984-03-29
987-65-4321	Kim	869-9999	1981-01-16
⋮			

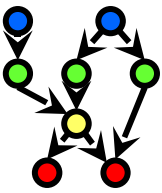




# FIRST NORMAL FORM

- 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 R is in first normal form if the domains of all attributes of R are atomic
- Non-atomic values complicate storage and encourage redundant (repeated) storage of data
  - ★ Example: Set of accounts stored with each customer, and set of owners stored with each account
  - ★ We assume all relations are in first normal form (and revisit this in Chapter 9)

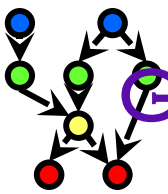




## FIRST NORMAL FORM (CONT'D)

- Atomicity is actually a property of how the elements of the domain are used.
  - ★ Example: Strings would normally be considered indivisible
  - ★ Suppose that students are given roll numbers which are strings of the form *CS0012* or *EE1127*
  - ★ If the first two characters are extracted to find the department, the domain of roll numbers is not atomic.
  - ★ Doing so is a bad idea: leads to encoding of information in application program rather than in the database.

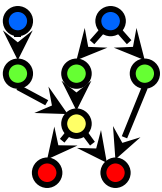




# GOAL – DEVISE A THEORY FOR THE FOLLOWING

- 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  $\{R_1, R_2, \dots, R_n\}$  such that
  - \* each relation is in good form
  - \* the decomposition is a lossless-join decomposition
- Our theory is based on:
  - \* functional dependencies  $\xrightarrow{1NF} \xrightarrow{2NF} \xrightarrow{3NF} \xrightarrow{BCNF}$
  - \* multivalued dependencies  $\xrightarrow{4NF}$

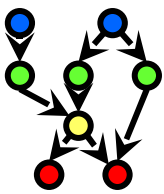




# FUNCTIONAL DEPENDENCIES

- 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

$$\alpha \subseteq R \text{ and } \beta \subseteq R$$

- The functional dependency

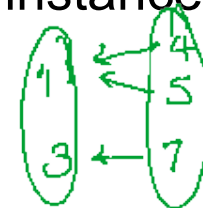
$$\alpha \rightarrow \beta$$

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  $\alpha$ , they also agree on the attributes  $\beta$ . That is,

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

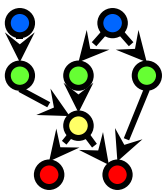
- Example: Consider  $r(A, B)$  with the following instance of  $r$ .

1	5
3	7



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





# FUNCTIONAL DEPENDENCIES (CONT.)

- $K$  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  $\alpha \subset K$ ,  $\alpha \rightarrow R$  } *minimal superkey*
- Functional dependencies allow us to express constraints that cannot be expressed using superkeys. Consider the schema:

*bor\_loan = (customer\_id, loan\_number, amount).*

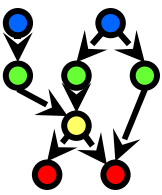
We expect this functional dependency to hold:

*loan\_number → amount*

but would not expect the following to hold:

*amount → customer\_name*

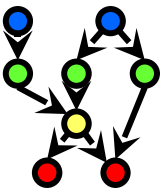




# USE OF FUNCTIONAL DEPENDENCIES

- We use functional dependencies to:
  - ★ test relations to see if they are legal under a given set of functional dependencies.
    - ⇒ If a relation  $r$  is legal under a set  $F$  of functional dependencies, we say that  $r$  satisfies  $F$ .
  - ★ 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  $F$ .
- Note: A specific instance  <sup>$r$</sup>  of a relation schema  <sup>$R$</sup>  may satisfy a functional dependency even if the functional dependency does not hold on all legal instances.
  - ★ For example, a specific instance of *loan* may, by chance, satisfy *amount* → *customer\_name*.

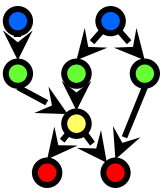




# FUNCTIONAL DEPENDENCIES (CONT.)

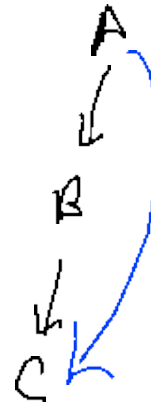
- A functional dependency is trivial if it is satisfied by all instances of a relation
  - ★ Example:
    - ⇒  $customer\_name, loan\_number \rightarrow customer\_name$
    - ⇒  $customer\_name \rightarrow customer\_name$
  - ★ In general,  $\alpha \rightarrow \beta$  is trivial if  $\beta \subseteq \alpha$

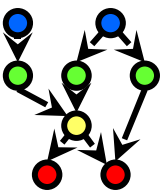




# CLOSURE OF A SET OF FUNCTIONAL DEPENDENCIES

- 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  $F^+$ .
- $F^+$  is a superset of  $F$ .





# BOYCE-CODD NORMAL FORM

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:

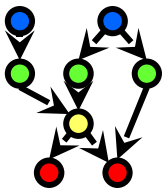
- $\alpha \rightarrow \beta$  is trivial (i.e.,  $\beta \subseteq \alpha$ )
- $\alpha$  is a superkey for  $R$

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





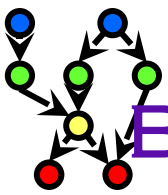
# DECOMPOSING A SCHEMA INTO BCNF

- 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,
    - \*  $\alpha = \text{loan\_number}$
    - \*  $\beta = \text{amount}$and  $\text{bor\_loan}$  is replaced by
    - \*  $(\alpha \cup \beta) = (\text{loan\_number}, \text{amount})$
    - \*  $(R - (\beta - \alpha)) = (\text{customer\_id}, \text{loan\_number})$

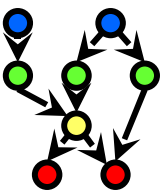




# BCNF AND DEPENDENCY PRESERVATION

- 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:

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

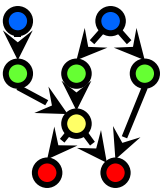
at least one of the following holds:

- ★  $\alpha \rightarrow \beta$  is trivial (i.e.,  $\beta \in \alpha$ )
- ★  $\alpha$  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).

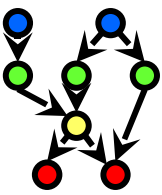




# GOALS OF NORMALIZATION

- 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?

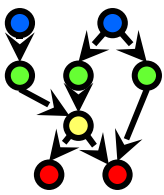
- There are database schemas in BCNF that do not seem to be sufficiently normalized
- Consider a database

*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

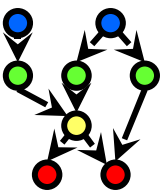
*classes*

- 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

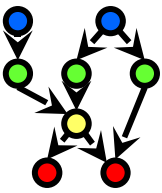
*teaches*

<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.

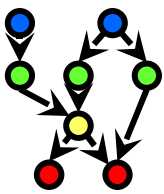




# FUNCTIONAL-DEPENDENCY THEORY

- 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



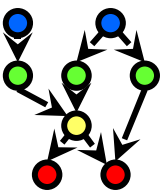


# CLOSURE OF A SET OF FUNCTIONAL DEPENDENCIES

- 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).







## 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  $F^+$

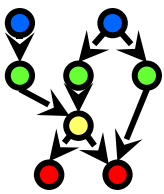
**if**  $f_1$  and  $f_2$  can be combined using transitivity

**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



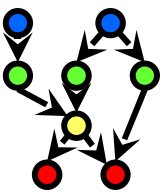


# CLOSURE OF FUNCTIONAL DEPENDENCIES (CONT.)

- 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.



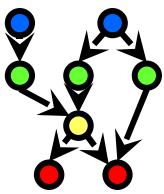


# CLOSURE OF ATTRIBUTE SETS

- Given a set of attributes  $\alpha$ , define the *closure* of  $\alpha$  under  $F$  (denoted by  $\alpha^+$ ) as the set of attributes that are functionally determined by  $\alpha$  under  $F$
- Algorithm to compute  $\alpha^+$ , the closure of  $\alpha$  under  $F$

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

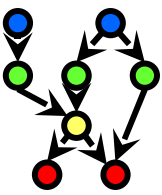




## EXAMPLE OF ATTRIBUTE SET CLOSURE

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



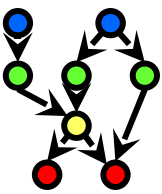


# USES OF ATTRIBUTE CLOSURE

There are several uses of the attribute closure algorithm:

- Testing for superkey:
  - ★ To test if  $\alpha$  is a superkey, we compute  $\alpha^+$ , and check if  $\alpha^+$  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  $\alpha^+$  by using attribute closure, and then check if it contains  $\beta$ .
  - ★ Is a simple and cheap test, and very useful
- Computing closure of  $F$ 
  - ★ For each  $\gamma \subseteq R$ , we find the closure  $\gamma^+$ , 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

