EDAYATHANGUDY G.S.PILLAY ARTS AND SCIENCE COLLEGE
Affiliated to Bharathidhasan university, Tiruchirappalli
(NAAC Accredited with "A"Grade)
Nagapattinam--611002

Subjectcode:16SCCCA4

Subject: DataBase System

Unit V:

Relational database design: Features of good Relation design - Atomic Domains and first normal form - Decomposition using functional dependencies – Functional-Dependencies theory - Decomposition using functional dependencies - Decomposition using Multivaluted dependencies – More Normal forms – Database-Design process

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 MultivaluedDependencies
- MoreNormal Form
- Database-Design Process
- Modeling Temporal Data

The Banking Schema

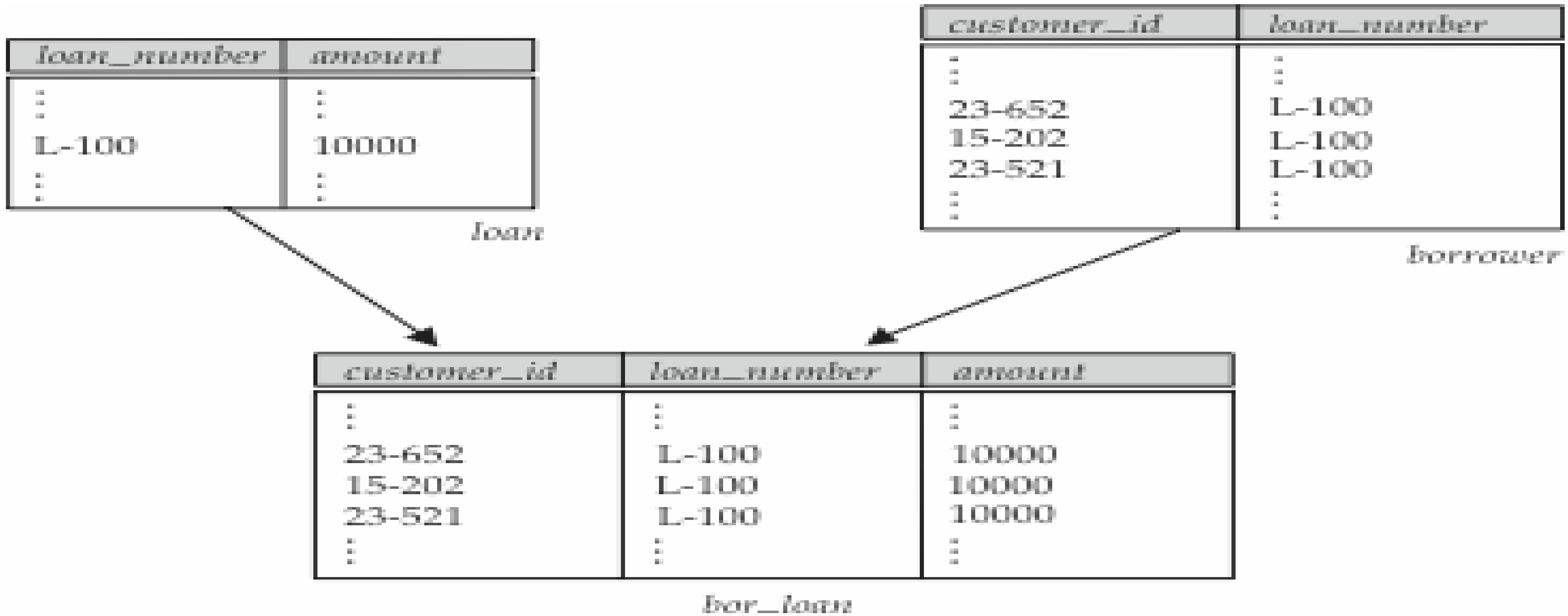
- *branch*= (*branch_name*, *branch_city*, *assets*)
- *customer*= (*customer_id*, *customer_name*, *customer_street*, *customer_city*)
- *loan*= (*loan_number*, *amount*)
- *account*= (*account_number*, *balance*)

- *employee*= (*employee_id*. *employee_name*, *telephone_number*, *start_date*)
- *dependent_name*= (*employee_id*, *dname*)
- *account_branch*= (*account_number*, *branch_name*)
- *loan_branch*= (*loan_number*, *branch_name*)
- *borrower*= (*customer_id*, *loan_number*)
- *depositor*= (*customer_id*, *account_number*)
- *cust_banker*= (*customer_id*, *employee_id*, *type*)
- *works_for*= (*worker_employee_id*, *manager_employee_id*)
- *payment*= (*loan_number*, *payment_number*, *payment_date*, *payment_amount*)
- *savings_account*= (*account_number*, *interest_rate*)
- *checking_account*= (*account_number*, *overdraft_amount*)

Combine Schemas?

- Suppose we combine *borrower*and *loan*to get

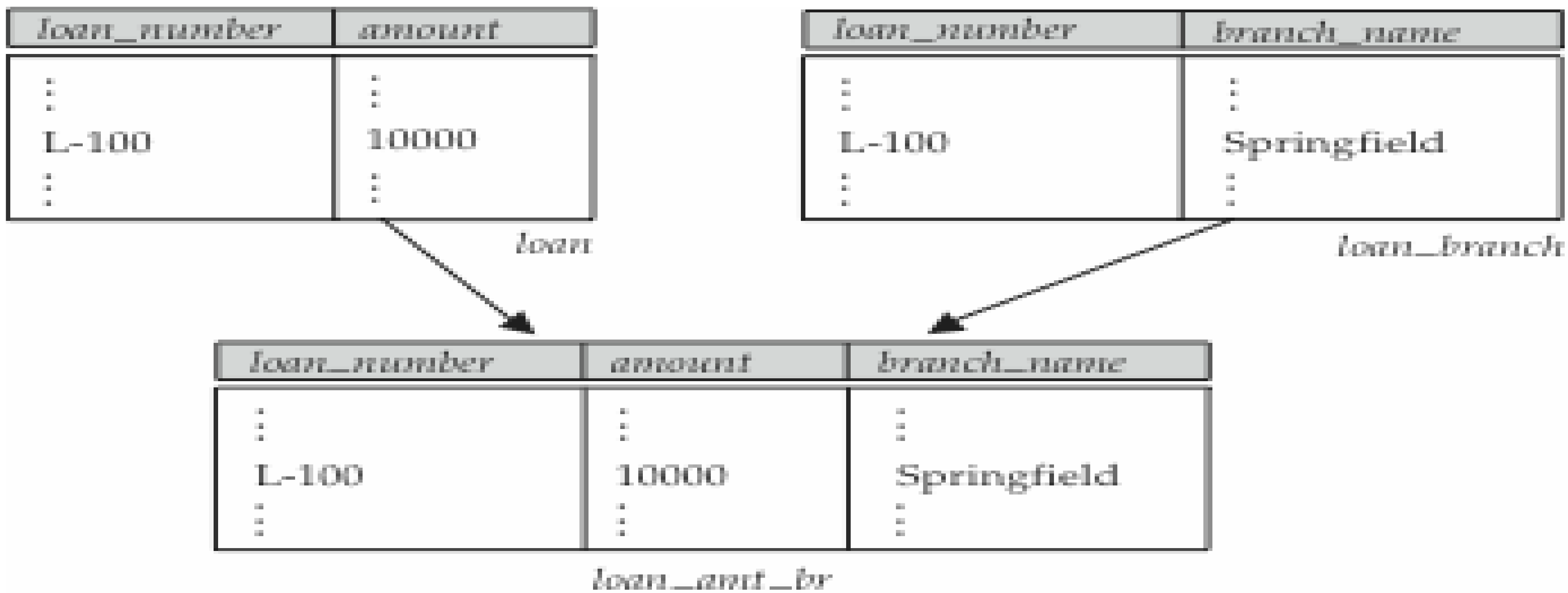
  bor_loan= (*customer_id*, *loan_number*, *amount*)
- Result is possible repetition of information (L-100 in example below)



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)



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**:

loan_number→*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 lossydecomposition.

A Lossy Decomposition

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

employee

employee_id	*employee_name*
⋮	
123-45-6789	Kim
987-65-4321	Kim
⋮	

employee_name	*telephone_number*	*start_date*
⋮		
Kim	882-0000	1984-03-29
Kim	869-9999	1981-01-16
⋮		

⋈

employee_id	*employee_name*	*telephone_number*	*start_date*
⋮			
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 atomicif 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 formif 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 Cont'd) 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 $\{R_1, R_2, ..., R_n\}$ such that
  - each relation is in good form
  - the decomposition is a lossless-join decomposition
- Our theory is based on:
  - functional dependencies
  - multivalued dependencies

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

  $\alpha \subseteq R$ *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, $A \to B$ does **NOT** hold, but $B \to A$ does hold.

Functional Dependencies (Cont.)

- K is a superkey for relation schema R if and only if $K \to R$
- K is a candidate key for R if and only if
  - $K \to R$, and
  - for no $\alpha \subset K$, $\alpha \to R$
- 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 of a relation schema 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* $\rightarrow$ *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* → *customer_name*
- *customer_name* → *customer_name*

- In general, α → β is trivial if β ⊆ α

Closure of a Set of Functional Dependencies

- Given a set *F* of functional dependencies, there are certain other functional dependencies that are logically implied by *F*.
- For example: If *A* → *B* and *B* → *C*, then we can infer that *A* → *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 Boyce-Codd Normal Form

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

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:

Example schema *not* in BCNF:

bor_loan = (*customer_id, loan_number, amount*)

because *loan_number* → *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 α →β causes a violation of BCNF.

We decompose *R* into:

- (α U β)
- (*R*-(β-α))

- In our example,
- ↑= *loan_number*
- ↓=*amount*

and *bor_loan*is replaced by

- (α U β) = (*loan_number, amount*)
- (*R*-(β-α)) = (*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, knownas *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:
- ↑❼↓is trivial (i.e., $\beta \in \alpha$)
- ↑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 $\{R_1, R_2, ..., R_n\}$ 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 $(c, t, b) \in classes$ 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.)

course	*teacher*	*book*
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:

course	*teacher*
Database database database operating systems operating systems	Avi Hank Sudarshan Avi Jim

Teaches

course	*book*
Database database database operating systems operating systems	DB Concepts Ullman OS Concepts Shaw

Text

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

Functional Functional-Dependency Theory

- We now consider the formal theory that tells us which functionaldependencies 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 allfunctional 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* →*B*

A →*C*

CG →*H*

CG →*I*

B →*H*}

- some members of F^+
- *A* →*H*
  - by transitivity from *A* →*B and B* →*H*
- *AG* →*I*
  - by augmenting *A* →*C* with G, to get *AG* →*CG* and then transitivity with *CG* →*I*

- *CG* →*HI*
  - by augmenting *CG* →*I* to infer *CG* →CG*I,* and augmenting of *CG* →*H* to infer*CGI* →*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 *a*nd $\alpha \rightarrow \gamma$ 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 *a*nd $\gamma\beta \rightarrow \delta$ 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 := α;
while (changes to *result*) **do**
    for each β → γ **in** *F* **do**

begin
if $\beta \subseteq$ *result* **then** *result* := *result* $\cup \gamma$
end

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

Extraneous Attributes

- Consider a set F of functional dependencies and the functional dependency $\alpha \rightarrow \beta$ in F.
- Attribute A is extraneous in α if $A \in \alpha$
and F logically implies $(F - \{\alpha \rightarrow \beta\}) \cup \{(\alpha - A) \rightarrow \beta\}$.
- Attribute A is extraneous in β if $A \in \beta$
and the set of functional dependencies
$(F - \{\alpha \rightarrow \beta\}) \cup \{\alpha \rightarrow (\beta - A)\}$ logically implies F.
- *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 $\{A \rightarrow C, AB \rightarrow C\}$ logically implies $A \rightarrow C$ (I.e. the result of dropping B from $AB \rightarrow C$).
- Example: Given $F = \{A \rightarrow C, AB \rightarrow CD\}$
- C is extraneous in $AB \rightarrow CD$ since $AB \rightarrow C$ 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 β; if it does, A is extraneous in α
- 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. check that α^+ contains A; if it does, A is extraneous in β

Computing a Canonical Cover

- $R = (A, B, C)$
$F = \{A \rightarrow BC$
$B \rightarrow C$
$A \rightarrow B$
$AB \rightarrow C\}$
- 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 $AB \rightarrow C$
- Check if the result of deleting A from $AB \rightarrow C$ is implied by the other dependencies
  - Yes: in fact, $B \rightarrow C$ is already present!
- Set is now $\{A \rightarrow BC, B \rightarrow C\}$
- C is extraneous in $A \rightarrow BC$

- Check if $A \rightarrow C$ is logically implied by $A \rightarrow B$ and the other dependencies
    - Yes: using transitivity on $A \rightarrow B$ and $B \rightarrow C$.
    - –Can use attribute closure of A in more complex cases
- The canonical cover is:

$A \rightarrow B$

$B \rightarrow C$

Lossless Lossless-join Decomposition

- For the case of $R = (R_1, R_2)$, we require that for all possible relations r on schema

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

- A decomposition of R into R_1 and R_2 is lossless join if and only if atleast one of the following dependencies is in F^+:
- $R_1 \cap R_2 \rightarrow R_1$
- $R_1 \cap R_2 \rightarrow R_2$

Example

- $R = (A, B, C) F = \{A \rightarrow B, B \rightarrow C)$
- Can be decomposed in two different ways
- $R_1 = (A, B), R_2 = (B, C)$

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

Dependency Preservation

- Let F_i be the set of dependencies F^+ that include only attributes in R_i.
  - A decomposition is dependency preserving, if

$$(F_1 \cup F_2 \cup \ldots \cup F_n)^+ = F^+$$

  - If it is not, then checking updates for violation of functional dependencies may equire computing joins, which is expensive.

Testing for Dependency Preservation

- To check if a dependency $\alpha \rightarrow \beta$ is preserved in a decomposition of R into R_1, R_2, …, R_n we apply the following test (with attribute closure done with respect to F)
- *result* = α
while (changes to *result*) do
for each R_i in the decomposition
$t = (result \cap R_i)^+ \cap R_i$
$result = result \cup t$
- If *result* contains all attributes in β, then the functional dependency
$\alpha \rightarrow \beta$ is preserved.
- We apply the test on all dependencies in F to check if a decomposition is dependency preserving
- This procedure takes polynomial time, instead of the exponential time required to compute F^+ and $(F_1 \cup F_2 \cup \ldots \cup F_n)^+$

Testing forBCNF

- To check if a non-trivial dependency $\alpha \rightarrow \beta$causes a violation of BCNF

  1. compute α^+(the attribute closure of α), and
  2. verify that it includes all attributes of *R*, that is, it is a superkey of *R*.

- Simplified test: To check if a relation schema *R*is in BCNF, it suffices to check only the dependencies in the given set *F*for violation of BCNF, rather than checking all dependencies in F^+.
- 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.
- However, using only *F*is incorrectwhen testing a relation in a decomposition of R
- Consider *R* =(*A, B, C, D, E*), with *F*= { *A* →*B, BC* →*D*}
  - Decompose *R*into R_1 =(*A,B*) and R_2 =(*A,C,D, E*)
  - Neither of the dependencies in *F*contain only attributes from (*A,C,D,E*) so we might be mislead into thinking R_2satisfies BCNF.
  - In fact, dependency *AC*→*D*in F^+shows $R_2$is not in BCNF.