RAWDATA
Section 1

Relational Database Design (Normalization)

Henrik Bulskov & Troels Andreasen

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

A schema with problems

- The schema below store info about instructors, their department and the location and budget of these
  - *this is an example of bad design*
  - *so what is bad here?*

inst_dept

ID	*name*	*salary*	*dept_name*	*building*	*budget*
22222	Einstein	95000	Physics	Watson	70000
12121	Wu	90000	Finance	Painter	120000
32343	El Said	60000	History	Painter	50000
45565	Katz	75000	Comp. Sci.	Taylor	100000
98345	Kim	80000	Elec. Eng.	Taylor	85000
76766	Crick	72000	Biology	Watson	90000
10101	Srinivasan	65000	Comp. Sci.	Taylor	100000
58583	Califieri	62000	History	Painter	50000
83821	Brandt	92000	Comp. Sci.	Taylor	100000
15151	Mozart	40000	Music	Packard	80000
33456	Gold	87000	Physics	Watson	70000
76543	Singh	80000	Finance	Painter	120000

What About Smaller Schemas?

- *what's bad is:* ***repetition of information*** *(redundancy)*

- What to do to avoid this?
  - decompose the schema
- How do we know a "good" way to split up (**decompose**) a schema?
- E.g. whether splitting *inst_dept* into *instructor* and *department* is a good decomposition?

- Notice
  - "if there were a schema (*dept_name, building, budget*), then *dept_name* would be a candidate key"
- This can be derived by identifying, what we call, a **functional dependency**:
  $dept\_name \rightarrow building, budget$

- Problem:
  - In *inst_dept*, because *dept_name* is not a candidate key, the building and budget of a department may have to be repeated
    - if dept_name is repeated, then so is building and budget
    - but only the repetition of building and budget is redundancy!
  - This indicates the need to decompose *inst_dept*

Decomposition without loss of information

inst_dept

ID	name	salary	dept_name	building	budget
22222	Einstein	95000	Physics	Watson	70000
12121	Wu	90000	Finance	Painter	120000
32343	El Said	60000	History	Painter	50000
45565	Katz	75000	Comp. Sci.	Taylor	100000
98345	Kim	80000	Elec. Eng.	Taylor	85000
76766	Crick	72000	Biology	Watson	90000
10101	Srinivasan	65000	Comp. Sci.	Taylor	100000
58583	Califieri	62000	History	Painter	50000
83821	Brandt	92000	Comp. Sci.	Taylor	100000
15151	Mozart	40000	Music	Packard	80000
33456	Gold	87000	Physics	Watson	70000
76543	Singh	80000	Finance	Painter	120000

$\Pi_{ID,\ name,\ dept\_name,\ salary}(inst\_dept)$

ID	name	dept_name	salary
10101	Srinivasan	Comp. Sci.	65000
12121	Wu	Finance	90000
15151	Mozart	Music	40000
22222	Einstein	Physics	95000
32343	El Said	History	60000
33456	Gold	Physics	87000
45565	Katz	Comp. Sci.	75000
58583	Califieri	History	62000
76543	Singh	Finance	80000
76766	Crick	Biology	72000
83821	Brandt	Comp. Sci.	92000
98345	Kim	Elec. Eng.	80000

$\Pi_{dept\_name,\ building,\ budget}(inst\_dept)$

dept_name	building	budget
Biology	Watson	90000
Comp. Sci.	Taylor	100000
Elec. Eng.	Taylor	85000
Finance	Painter	120000
History	Painter	50000
Music	Packard	80000
Physics	Watson	70000

ID	name	salary	dept_name	building	budget
22222	Einstein	95000	Physics	Watson	70000
12121	Wu	90000	Finance	Painter	120000
32343	El Said	60000	History	Painter	50000
45565	Katz	75000	Comp. Sci.	Taylor	100000
98345	Kim	80000	Elec. Eng.	Taylor	85000
76766	Crick	72000	Biology	Watson	90000
10101	Srinivasan	65000	Comp. Sci.	Taylor	100000
58583	Califieri	62000	History	Painter	50000
83821	Brandt	92000	Comp. Sci.	Taylor	100000
15151	Mozart	40000	Music	Packard	80000
33456	Gold	87000	Physics	Watson	70000
76543	Singh	80000	Finance	Painter	120000

$$inst\_dept = \Pi_{ID,\ name,\ dept\_name,\ salary}(inst\_dept) \bowtie \Pi_{dept\_name,\ building,\ budget}(inst\_dept)$$

Example of Lossless-Join Decomposition

- **Lossless join decomposition**
- Decomposition of $R = (A, B, C)$ into

  $R_1 = (A, B) \quad R_2 = (B, C)$

A	B	C
α	1	A
β	2	B

r

A	B
α	1
β	2

$\Pi_{A,B}(r)$

B	C
1	A
2	B

$\Pi_{B,C}(r)$

$\Pi_{A,B}(r) \bowtie \Pi_{B,C}(r)$

A	B	C
α	1	A
β	2	B

- So here we have that

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

Example of Lossy Decomposition

- **Decomposition that is not a lossless join decomposition**
- Decomposition of $R = (A, B, C)$ into

$$R_1 = (A, B) \qquad R_2 = (B, C)$$

A	B	C
α	1	A
β	1	B

r

A	B
α	1
β	1

$\Pi_{A,B}(r)$

B	C
1	A
1	B

$\Pi_{B,C}(r)$

$\Pi_{A,B}(r) \bowtie \Pi_{B,C}(r)$

A	B	C
α	1	A
β	1	B
α	1	A
β	1	B

- So in this case

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

Another Lossy Decomposition

- So not all decompositions are good.
- We loose information by choosing a bad decomposition of ***employee***

ID	*name*	*street*	*city*	*salary*
⋮				
57766	*Kim*	*Main*	*Perryridge*	*75000*
98776	*Kim*	*North*	*Hampton*	*67000*
⋮				

employee

ID	*name*
⋮	
57766	*Kim*
98776	*Kim*
⋮	

name	*street*	*city*	*salary*
⋮			
Kim	*Main*	*Perryridge*	*75000*
Kim	*North*	*Hampton*	*67000*
⋮			

natural join

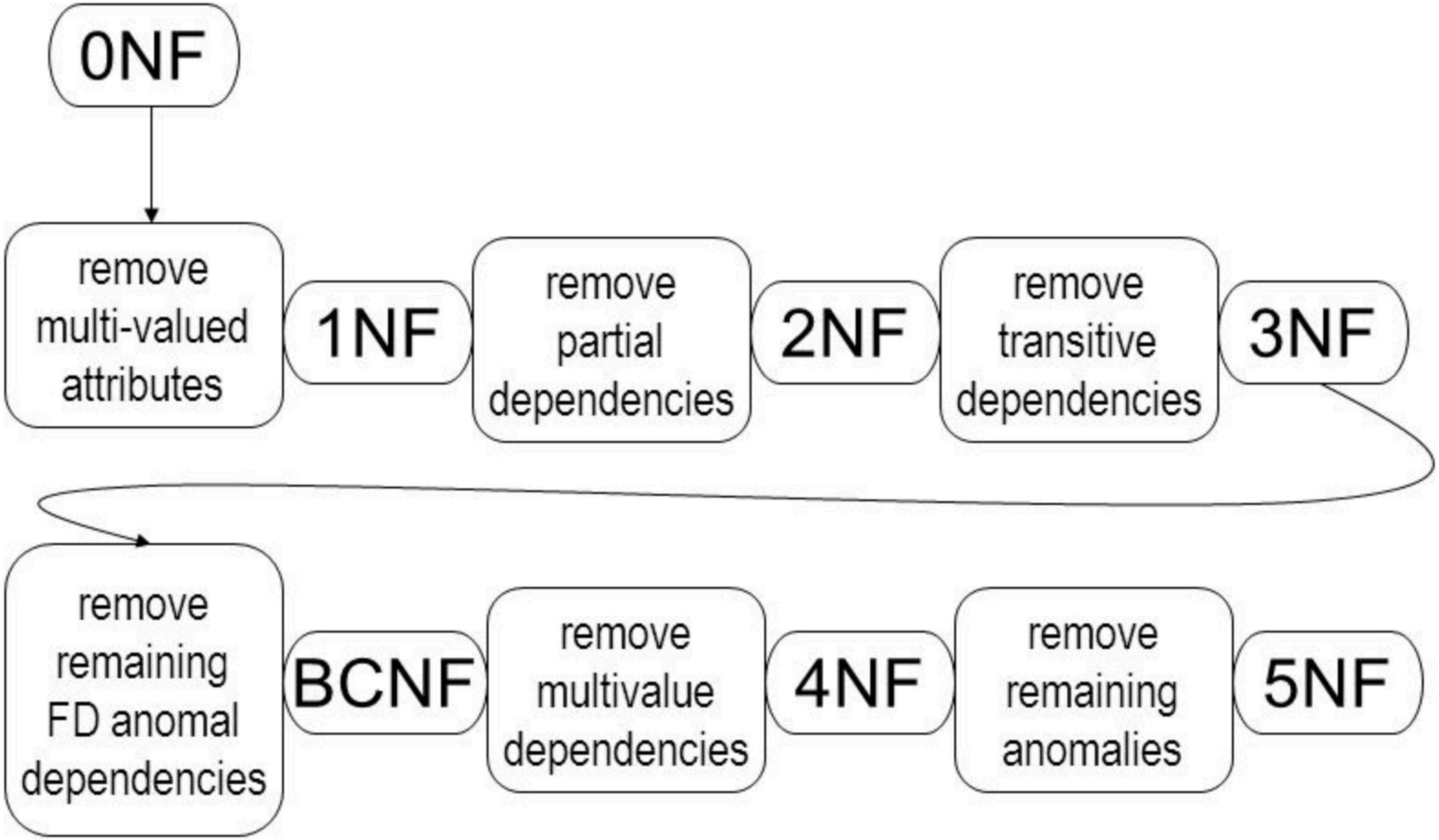
ID	*name*	*street*	*city*	*salary*
⋮				
57766	*Kim*	*Main*	*Perryridge*	*75000*
57766	*Kim*	*North*	*Hampton*	*67000*
98776	*Kim*	*Main*	*Perryridge*	*75000*
98776	*Kim*	*North*	*Hampton*	*67000*
⋮				

Normalization

- Database normalization, the process of
  - changing the database schema to reduce data redundancy
  - simplifying the design of a database by decomposing this
  - Bringing relational schemas to (increasingly restrictive) Normal forms

- Normal forms
  - **1NF: First Normal Form**
  - 2NF: Second Normal Form
  - 3NF: Third Normal Form
  - **BCNF: Boyce-Codd Normal form**
  - 4NF: Fourth Normal Form
  - 5NF: Fifth Normal Form

Normalization



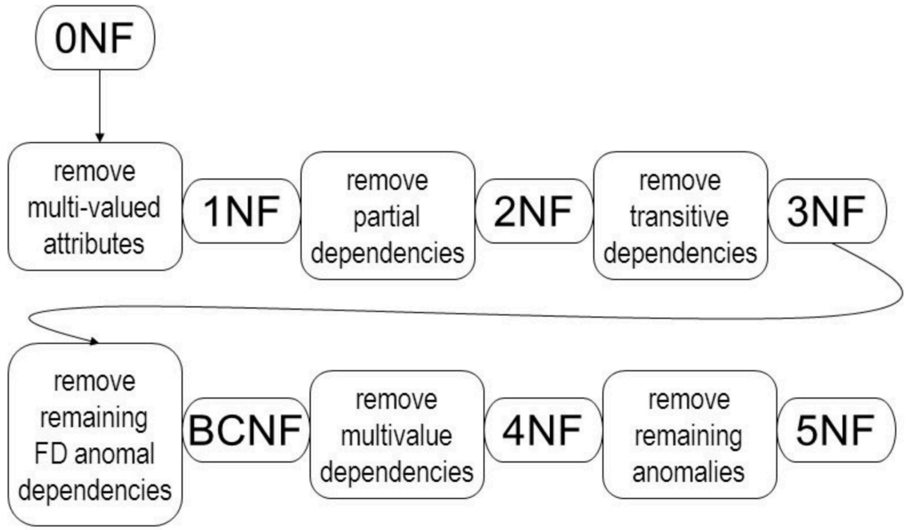
Normalization

- The order shows
  - increasingly more restrictive normal forms
  - thus, we can go backwards, and say for instance
    - if R in BCNF then R is in 3NF
    - if R in 3NF then R is in 2NF
    - if R in 2NF then R is in 1NF

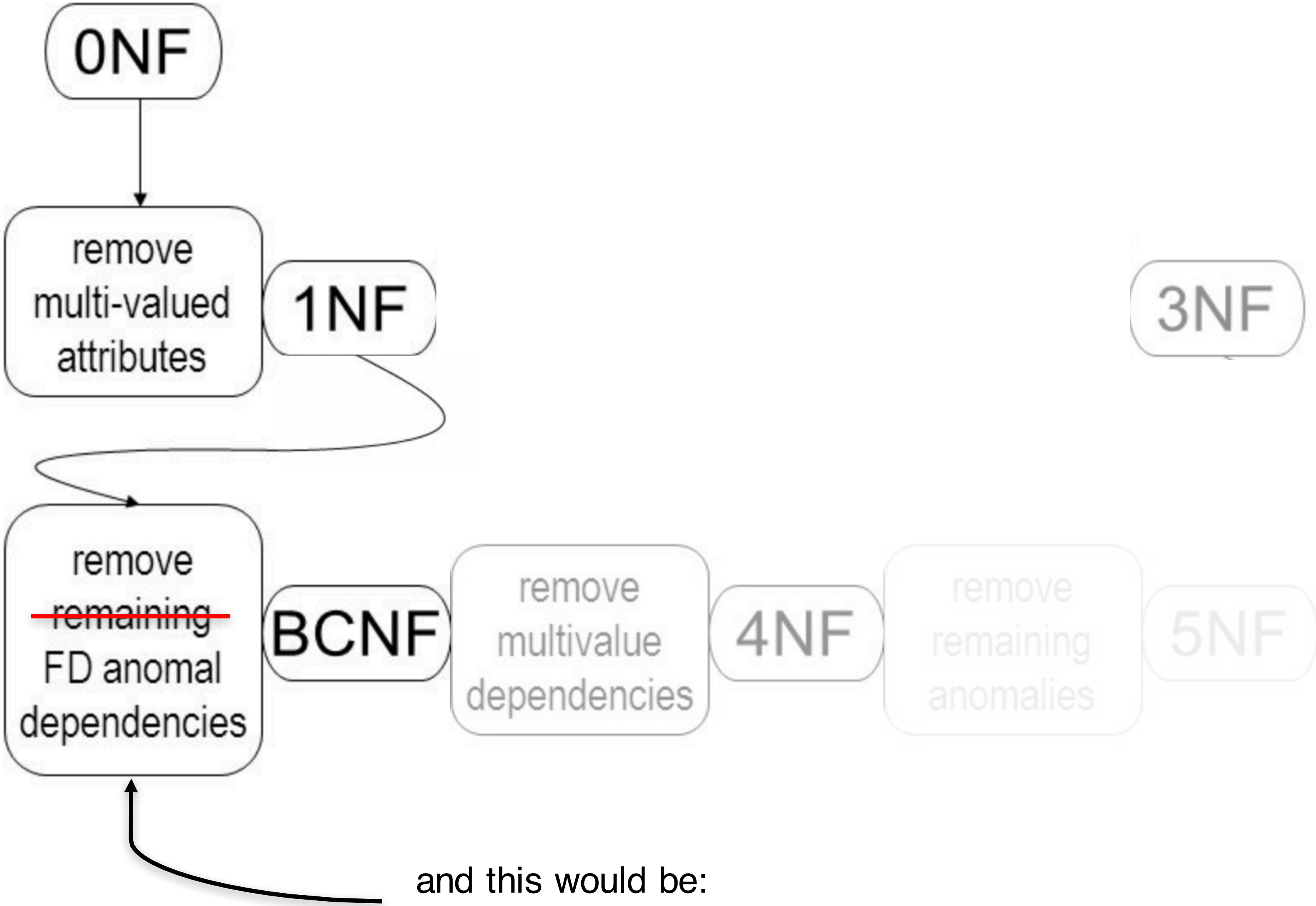
This is a mistake

- However
  - if you google NF's (normal), you'll find some explanations like this:
    - 2NF is 1NF + something more
    - 3NF is 2NF + something more
    - **BCNF is 3NF + something more**
  - **BCNF is NOT dependant on 3NF**

- observe
  - rather than
    - 3NF is 2NF + something more
  - the DSC book defines 3NF independently (or as BCNF + a relaxation)



Normalization
- **our main focus**



and this would be:
- dependencies, where the left hand side is not a superkey

First Normal Form

- A domain is **atomic** if its elements are considered to be indivisible units
  - Examples of **non-atomic** domains:
    - Set of values like: phone numbers,
    - a composition like: an address
    - 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**

- Why 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

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 should be 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

- functional dependency, example

  dept_name $\rightarrow$ *building, budget*

  – can be read like:
    - *dept_name* determines *building* and *budget*

  meaning
  – *dept_name* determines the value of *building*
    - for a given value of *dept_name* there is only one value for *building*
  – *dept_name* determines the value of *budget*
    - for a given value of *dept_name* there is only one value for *budget*

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 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 (Cont.)

- Functional dependencies allow us to express constraints

- Key constraints:
  - 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 (uniqueness)
    - for no $K' \subset K$, $K' \rightarrow R$ (minimality)

- Other constraints
  - Consider the schema:

    inst_dept (<u>*ID,*</u> *name, salary, dept_name, building, budget*).

    We expect these functional dependencies to hold:

    $dept\_name \rightarrow building$ and

    $ID \rightarrow building$

    but would not expect the following to hold:

    $dept\_name \rightarrow salary$

Trivial Functional Dependency

- *A* functional dependency is **trivial** if it is satisfied by all instances of a relation
  - Example*:*
    - *ID, name* $\rightarrow$ *ID*
    - *name* $\rightarrow$ *name*
  - In general, $\alpha \rightarrow \beta$ is trivial if $\beta \subseteq \alpha$

Use of Functional Dependencies

- So, suppose we have
  - a set of functional dependencies ***F***
  - a relation ***R***
  - an instance ***r*** of ***R***

- We say that
  - ***r* satisfies *F*** : on the instance ***r*** all dependencies in ***F*** hold (incident)
  - ***F* holds on *R*** : all legal instances on ***R*** satisfies ***F*** (by "law")

- We use the functional dependencies ***F*** to:
  1) test a specific instance ***r*** of the relation ***R*** to see if it is legal (legal means: that ***r*** satisfies ***F***)
  2) specify constraints on the set of legal instances of a relation ***R***

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 $A \rightarrow B$ and $B \rightarrow C$, then we can infer that $A \rightarrow C$

- *Closure* $\boldsymbol{F^+}$ of F
  - The set of **all** functional dependencies logically implied by F
  - Denoted by $\boldsymbol{F^+}$
  - $\boldsymbol{F^+}$ 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^+

$$\alpha \rightarrow \beta$$

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:

instr_dept (*<u>ID,</u> name, salary, dept_name, building, budget*)

because *dept_name* $\rightarrow$ *building, budget*
holds on *instr_dept,* but *dept_name* 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:
  - (α U β)
  - (*R* - (β - α))

- In our example, ***inst_dept***
  - α = *dept_name*
  - β = *building, budget*

  the original schema is replaced by two schemas:
  - (α U β) = (*dept_name, building, budget*)
  - (*R* - (β - α)) = (*ID, name, salary, dept_name*)

  We may choose meaningful names for these two new schemas such as: ***instructor*** and ***department*** respectively

BCNF and Dependency Preservation

- Constraints, including functional dependencies, are costly to check in practice unless they pertain to only one relation

- *Dependency preserving* decomposition
  - All dependencies still pertain to only one relation after decomposition
  - it is thus sufficient to test dependencies on each individual relation of a decomposition.

- Because it is not always possible to achieve both BCNF and dependency preservation, we may 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., $\beta \in \alpha$)
  - 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.

Normalization – How to

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

How good is BCNF?

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

  inst_info (ID, child_name, phone)

  – may have more than one phone and can have multiple children

ID	*child_name*	*phone*
99999	David	512-555-1234
99999	David	512-555-4321
99999	William	512-555-1234
99999	William	512-555-4321

inst_info

- There are no non-trivial functional dependencies and therefore the relation is in BCNF
- Insertion anomalies – i.e., if we add a phone 981-992-3443 to 99999, we need to add two tuples

  (99999, David, 981-992-3443)
  (99999, William, 981-992-3443)

How good is BCNF? (Cont.)

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

inst_child

ID	*child_name*
99999	David
99999	William

inst_phone

ID	*phone*
99999	512-555-1234
99999	512-555-4321

This suggests the need for higher normal forms, such as Fourth Normal Form (4NF).

Functional-Dependency Theory

- Problem:
  - We need to know what functional dependencies that hold

- We now consider the formal theory that tells us which functional dependencies are implied logically by a given set of functional dependencies.

- We then consider algorithms to
  - generate lossless decompositions into BCNF and 3NF
  - test if a decomposition is dependency-preserving

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.
  Example:
  - If $A \rightarrow B$ and $B \rightarrow C$, then we can infer that $A \rightarrow C$
  - so $A \rightarrow C$ is logically implied by the set $\{A \rightarrow B, B \rightarrow C\}$
- The set of **all** functional dependencies logically implied by F is the **closure** of F.
- We denote the *closure* of F by F^{+}.

Closure of a Set of Functional Dependencies

- We can find F^{+}, the closure of F, by repeatedly 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)**

- Examples:
  - $\{ID\} \subseteq \{ID, name\}$ **so:** $ID, name \rightarrow ID$
  - $ID \rightarrow name$ **so:** $ID, salary \rightarrow name, salary$
  - $ID \rightarrow dept\_name$, and $dept\_name \rightarrow building$ **so:** $ID \rightarrow building$

- These rules are
  - **sound** (generate only functional dependencies that actually hold), and
  - **complete** (generate all functional dependencies that hold).

Closure of Functional Dependencies (Cont.)

- Additional rules:
  - If $\alpha \rightarrow \beta$ and $\alpha \rightarrow \gamma$ holds, then $\alpha \rightarrow \beta\gamma$ holds (**union**)
  - If $\alpha \rightarrow \beta\gamma$ holds, then $\alpha \rightarrow \beta$ and $\alpha \rightarrow \gamma$ holds (**decomposition**)
  - If $\alpha \rightarrow \beta$ and $\gamma\beta \rightarrow \delta$ holds, then $\alpha\gamma \rightarrow \delta$ holds (**pseudotransitivity**)

- These rules
  - can be inferred from Armstrong's axioms so they are redundant
  - they can, however, be convenient

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* $B \rightarrow H$
  - $AG \rightarrow I$
    - How?
  - $CG \rightarrow HI$
    - How?
- rules are
  - 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)**
  - If $\alpha \rightarrow \beta$ and $\alpha \rightarrow \gamma$ holds, then $\alpha \rightarrow \beta\gamma$ holds **(union)**
  - If $\alpha \rightarrow \beta\gamma$ holds, then $\alpha \rightarrow \beta$ and $\alpha \rightarrow \gamma$ holds **(decomposition)**
  - If $\alpha \rightarrow \beta$ and $\gamma\beta \rightarrow \delta$ holds, then $\alpha\gamma \rightarrow \delta$ holds **(pseudotransitivity)**

Procedure for Computing F^+

- Based on the three basic axioms (reflexivity, augmentation and transivity).
- 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

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)**

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

Closure of Attribute Sets

- We can derive
  - Closure of a Set of Functional Dependencies, however, we can also derive
  - Closure of a Set of attributes

- Given a set of attributes α, the ***closure*** α^+ of α **under** *F* is defined 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 $\beta \rightarrow \gamma$ **in** *F* **do**
        begin
          if $\beta \subseteq$ *result* **then** *result* := *result* $\cup\ \gamma$
        end

Example of Attribute Set Closure

- $R = (A, B, C, G, H, I)$
- $F = \{CG \rightarrow H$
  $CG \rightarrow I$
  $A \rightarrow B$
  $A \rightarrow C$
  $B \rightarrow H\}$
- $(AG)^+$
  1. $result = AG$
  2. $result = ABCG$ $(A \rightarrow C$ and $A \rightarrow B)$
     - Are we done?

- *Using attribute closure to test if, e.g., AG is a candidate key*
  1. Is AG a super key?
     1. Does $AG \rightarrow R$? (Is $(AG)^+ = R$)
  2. Is any subset of AG a superkey?
     1. Does $A \rightarrow R$? (Is $(A)^+ = R$)
     2. Does $G \rightarrow R$? (Is $(G)^+ = R$)

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 $\alpha^{+,}$ and check if α^{+} contains all attributes of R (example on previous slide)

- 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.
  - This is a simple and cheap test, and very useful

- Computing closure F^{+} of F (alternative)
  - For each $\gamma \subseteq R$, we find the closure γ^{+}, and for each subset $S \subseteq \gamma^{+}$, we output a functional dependency $\gamma \rightarrow S$.

Redundant dependencies / attributes

- Sets of functional dependencies may have redundant dependencies, that is: dependencies that can be inferred from the others
  - For example: $A \rightarrow C$ is redundant in: $\{A \rightarrow B, B \rightarrow C, A \rightarrow C\}$

- Also parts of a functional dependency may be redundant
  - E.g. (left): $\{A \rightarrow B, \quad B \rightarrow C, \quad AC \rightarrow D\}$ can be simplified to
    $\{A \rightarrow B, \quad B \rightarrow C, \quad A \rightarrow D\}$. (thus C is **extraneous**)
  - E.g. (right): $\{A \rightarrow B, \quad B \rightarrow C, \quad A \rightarrow CD\}$ can be simplified to
    $\{A \rightarrow B, \quad B \rightarrow C, \quad A \rightarrow D\}$ (thus C is **extraneous**)

- 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.

- Example: Given $F = \{A \rightarrow C, A\mathbf{B} \rightarrow C\}$
  - 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 $F = \{A \rightarrow C, AB \rightarrow \mathbf{C}D\}$
  - C is extraneous in $AB \rightarrow CD$ since $AB \rightarrow CD$ can be inferred even after deleting C

Testing if an Attribute is Extraneous using Attribute Closure

- 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

Canonical Cover

- A **canonical cover** for F is a set of dependencies F_c such that
  - F logically implies all dependencies in F_c, and
  - F_c logically implies all dependencies in F, and
  - No functional dependency in F_c contains an extraneous attribute, and
  - Each left side of functional dependency in F_c is unique.
- To compute a canonical cover for F:

  repeat
  Use the union rule to replace any dependencies in F
  $\alpha_1 \rightarrow \beta_1$ and $\alpha_1 \rightarrow \beta_2$ 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
  If an extraneous attribute is found, delete it from $\alpha \rightarrow \beta$
  until F does not change
- Note: Union rule may become applicable after some extraneous attributes have been deleted, so it has to be re-applied

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-join Decomposition

- To decompose R into (R_1, R_2), we require that for all possible relations r on schema R

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

- A decomposition of R into R_1 and R_2 is lossless join if at least 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

- $R_1 = (A, B), \quad R_2 = (B, C)$
  - Lossless-join decomposition:
    $$R_1 \cap R_2 = \{B\} \text{ and } B \rightarrow BC$$
  - Dependency preserving

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

Testing for BCNF

- To check if a non-trivial dependency $\alpha \to \beta$ 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, **simplified test using only *F* is incorrect when testing a relation in a decomposition of R**
  - Consider $R = (A, B, C, D, E)$, with $F = \{ A \to B, BC \to D\}$
    - 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_2 satisfies BCNF.
    - In fact, dependency $AC \to D$ in F^+ shows R_2 is not in BCNF.

Testing Decomposition for BCNF

- To check if a relation R_i in a decomposition of R is in BCNF,
  - **Either** test R_i for BCNF with respect to the **restriction** of F^+ to R_i (that is, all FDs in F^+ that contain only attributes from R_i)

  - **or** use the original set of dependencies F that hold on R, but with the following test:
    - for every set of attributes $\alpha \subseteq R_i$, check that α^+ (the attribute closure of α) either includes no attribute of $R_i - \alpha$, or includes all attributes of R_i.
  - If the condition is violated by some $\alpha \rightarrow \beta$ in F, the dependency

    $\alpha \rightarrow (\alpha^+ - \alpha) \cap R_i$

    can be shown to hold on R_i, and R_i violates BCNF.
  - We use above dependency to decompose R_i

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 $\alpha \rightarrow \beta$ be a nontrivial functional dependency that
        holds on R_i such that $\alpha \rightarrow R_i$ is not in F^+,
        and $\alpha \cap \beta = \emptyset$;
      result := (*result* – R_i) $\cup$ ($R_i - \beta$) $\cup$ (α, β);
    end
  else *done* := **true;**

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

Example of BCNF Decomposition

- $R = (A, B, C)$
  $F = \{A \rightarrow B$
  $B \rightarrow C\}$
  Key $= \{A\}$
- R is not in BCNF ($B \rightarrow C$ but B is not superkey)
- Decomposition
  - $R_1 = (B, C)$
  - $R_2 = (A,B)$

Overall Database Design Process

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

Denormalization for Performance

- May want to use non-normalized schema for performance
- For example, displaying *prereqs* along with *course_id,* and *title* requires join of *course* with *prereq*

- Alternative 1: Use denormalized relation containing attributes of *course* as well as *prereq* 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
  $course \bowtie prereq$
  - 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*), are used

  – *earnings_2004, earnings_2005, earnings_2006*, 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_2004, earnings_2005, earnings_2006*)
    - Also in BCNF, but also makes querying across years difficult and requires new attribute each year.

Second Normalform

- violated when a non-key field is a fact about a subset of a key (thus only relevant when the key is composite)
- Consider the following schema:

  R(<u>PART</u>, <u>WAREHOUSE</u>, QUANTITY, WAREHOUSE-ADDRESS)
- suppose WAREHOUSE-ADDRESS is a fact about the WAREHOUSE

  WAREHOUSE -> WAREHOUSE-ADDRESS

- Problem:
  - The warehouse address is repeated in every row that refers to the warehouse

- Decompose into

  R1(<u>PART</u>, <u>WAREHOUSE</u>, QUANTITY)

  R2(<u>WAREHOUSE</u>, WAREHOUSE-ADDRESS)

See William Kent: A Simple Guide to Five Normal Forms

Third Normalform

- violated when a non-key field is a fact about another non-key field
- Consider the following schema:

  R(<u>EMPLOYEE</u>, DEPARTMENT, LOCATION)
- suppose each department is located in one place, thus

  DEPARTMENT -> LOCATION

- Problem:
  - The department's location is repeated in each row of every employee assigned to that department

- Decompose into

  R1(<u>EMPLOYEE</u>, DEPARTMENT)

  R2(<u>DEPARTMENT</u>, LOCATION)

See William Kent: A Simple Guide to Five Normal Forms