Database Normalization
(Relational Database Design)

Motivation

- There are “better” and “worse” relational schemas.
- Example:

course-schema						
c_code	title	creditPoints	**Hours-week**	lecturer	studentId	grade
EECS339	‘DBMS’	1	3	4711	1001	D
EECS339	‘DBMS’	1	3	4711	1010	A
EECS339	‘DBMS’	1	3	4711	1007	P
EECS203	‘Intro CE’	2	6	3001	NULL	NULL

Redundant Information

Incomplete Information

- How can we judge the quality of relational schemas?

Database Design Goals

- Relational database design requires that we find a “good” collection of relation schemas. A bad design may lead to
  - Repetition of Information.
  - Inability to represent certain information.
- Design Goals:
  - Avoid redundant data
  - Ensure that relationships among attributes are represented.
  - Ability to insert/update/delete information without sacrificing any primary or foreign key constraints and without the need for (extensive) use of null values

Evils of Redundancy

- *Redundancy* is at the root of several problems associated with relational schemas:
  - redundant storage
  - **Insertion Anomaly:**
    Adding new rows forces user to create duplicate data or to use *null* values.
  - **Deletion Anomaly:**
    Deleting rows may cause a loss of data that would be needed for other future rows!
  - **Update Anomaly:**
    Changing data in a row forces changes to other rows because of duplication.

Anomalies Example

course-schema						
c_code	title	creditPoints	**Hours-week**	lecturer	studentId	grade
EECS339	'DBMS'	1	3	4711	1001	D
EECS339	'DBMS'	1	3	4711	1010	A
EECS339	'DBMS'	1	3	4711	1007	P
Comp3005	'Intro CE'	2	6	3001	NULL	NULL

Question – Is this a relation?

Answer – Yes: unique rows and no multivalued attributes

Question – What's the primary key?

Answer – Composite: c_code, sid (but then no NULL values allowed!)

Question – What happens with data modifications?

Anomalies in Previous Example

- Insertion Anomaly:
  - If another student enrols in EECS339, we have to re-enter the course information, causing duplication.
  - What if we want to insert a course where no students enrolled so far? We either cannot do it at all (PK!) or we get many *NULL* values.
- Deletion Anomaly:
  - If we delete all courses with 2 credit points, we loose the information about the weekly workload for courses with 2 points!
  - Or if composite PK, we cannot delete the last student in a course!
- Update Anomaly:
  - For changing, e.g., the *creditPoints* of one course, we have to update multiple tuples.

Why do these anomalies exist?

Because there are two entity types placed into one relation. This results in duplication and an unnecessary dependencies

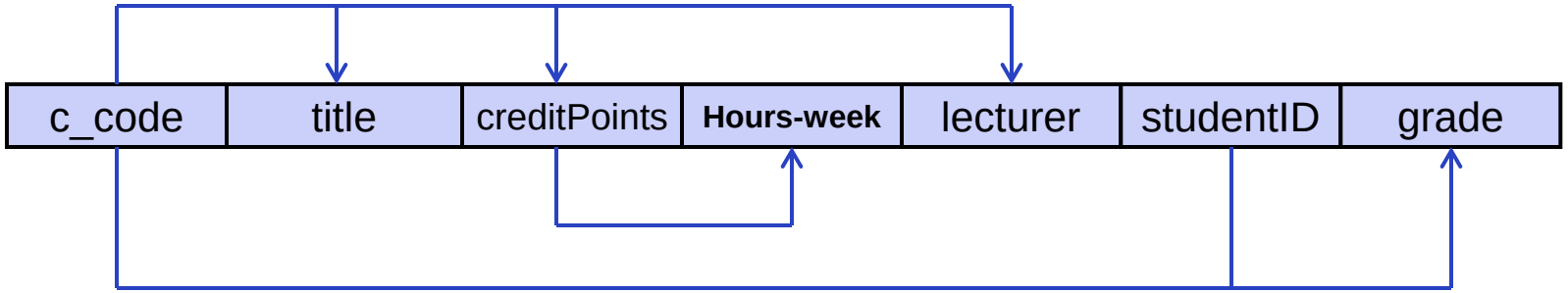
Functional Dependency (FD)

- Integrity constraints, in particular functional dependencies, can be used to identify schemas with such problems and to suggest refinements.
- **Functional Dependency**: The value of one attribute (the determinant) determines the value of another attribute
  - Intuitively: “If two tuples of a relation R agree on values in X, then they must also agree on the Y values.”
- We write $X \rightarrow Y$
  - “X (functionally) determines Y”
  - “Y is functionally dependent on X”

FD Example

- FDs in motivating example:
  - *c_code* → *title creditPoints lecturer*
  - *creditPoints* → *hours-week*
  - *c_code studentID* → *grade*

- Graphical notation:



- Which FD do not hold?

Formal Definition for FD

- Given a relation with schema $\boldsymbol{R}$, and two sets of attributes $X = \{X_1, \ldots, X_m\} \subseteq \boldsymbol{R}$ and $Y = \{Y_1, \ldots, Y_n\} \subseteq \boldsymbol{R}$.
  A **functional dependency (FD)** $X \rightarrow Y$ holds over relation schema $\boldsymbol{R}$ if, for every allowable instance R of $\boldsymbol{R}$:

$$\forall \; r, s \in R : \quad r.X = s.X \quad \Rightarrow \quad r.Y = s.Y$$

- A functional dependency $X \rightarrow Y$ *is said to be* ***trivial*** if $Y \subseteq X$

Some Remarks

- A FD is an assertion about the schema of a relation not about a particular instance.
  - It must be fulfilled by all allowable relations.
- If we look at an instance, we cannot tell for sure that a FD holds
  - The most insight we can gain by looking at a “typical” instance are “hints”…
  - We can however check if the instance violates some FD
- FDs must be identified based on the semantics of an application.

From FDs to Keys

- FDs can be used to identify schemas with problems and to suggest refinements.
- Main Idea: Only allow FDs of form of key constraints.
  - Each non-key field is functionally dependent on every candidate key

- Definition: **Superkey**
  Given a relation R with schema $\boldsymbol{R}$ and a set F of functional dependencies. A set of attributes $K \subseteq \boldsymbol{R}$ is a **superkey** for R if $K \rightarrow$ ***all attributes in R***

- Note that $K \rightarrow R$ does not require K to be minimal!
  - K is a **candidate key** if no real subset of K has above property.
  - One of the candidate keys will become the **primary key**

Reasoning about FDs

- Given some FDs, we can usually infer additional FDs:
  - Example:
    given *c_code* $\rightarrow$ *creditpoints* *and* *creditpoints* $\rightarrow$ *hours-week*
  - implies c_*code* $\rightarrow$ hours-week

- A FD *f* is *implied by* a set of FDs *F* if *f* holds whenever all FDs in *F* hold.

- F^+ : closure of *F* is the set of all FDs that are implied by *F*

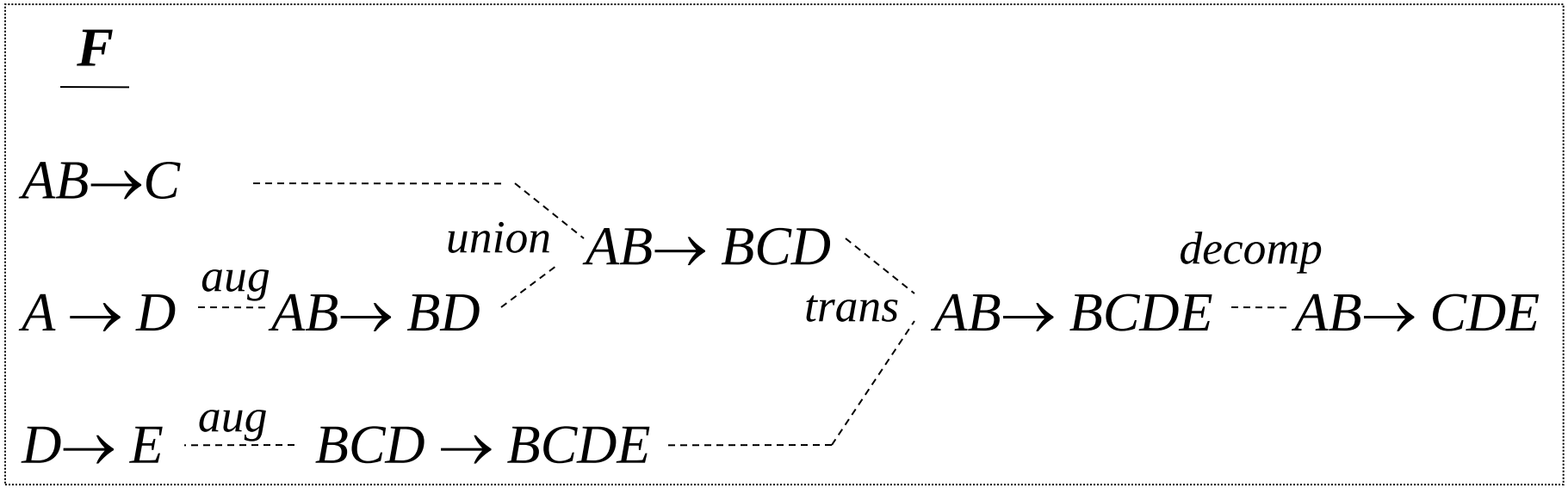
- **Armstrong's Axioms** (*X, Y, Z* are sets of attributes):

1. Reflexivity rule: If $X \subseteq Y$, then $Y \rightarrow X$
2. Augmentation rule: If $X \rightarrow Y$, then $XZ \rightarrow YZ$ for any Z
3. Transitivity rule: If $X \rightarrow Y$ and $Y \rightarrow Z$, then $X \rightarrow Z$

Reasoning about FDs (cont'd)

- Armstrong's Axioms are
  - *Sound*: they generate only FDs in F^+ when applied to a set F of FDs
  - *Complete*: repeated application of these rules will generate all FDs in the closure F^+
- A couple of additional rules (that follow from Armstrong's Axioms.):
  - Union rule: If $X \rightarrow Y$ and $X \rightarrow Z$, then $X \rightarrow YZ$
  - Decomposition rule: If $X \rightarrow YZ$, then $X \rightarrow Y$ and $X \rightarrow Z$
  - Pseudotransitivity rule: If $X \rightarrow Y$ and $SY \rightarrow Z$, then $XS \rightarrow Z$

Example: Generating F^+



Thus, $AB \rightarrow BD$, $AB \rightarrow BCD$, $AB \rightarrow BCDE$, and $AB \rightarrow CDE$ are all elements of F^+

Attribute Closure

- Calculating *attribute closure* leads to a more efficient way of checking for candidate keys

- The ***attribute closure*** of a set of attributes, X, with respect to a set of functional dependencies, $\boldsymbol{F}$, (denoted as $\boldsymbol{X^+}$) is the set of all attributes, A, such that $X \rightarrow A$

- *Attribute closure Algorithm*:
  - AttributeClosure(F, X):
    closure := X
    while (*closure changes*) **do**
      for each FD $Y \rightarrow Z$ **in** F **do**
        if $Y \subseteq$ *closure* **then** *closure* := *closure* $\cup$ Z

Example of attribute closure

StudentGrade (*SID*, *name*, *email*, *CID*, *grade*)

- *SID* $\rightarrow$ *name*, *email*
- *email* $\rightarrow$ *SID*
- *SID*, *CID* $\rightarrow$ *grade*

(Not a good design !)

Example of computing closure (ctd)

- F includes:
  - *SID* → *name*, *email*
  - *email* → *SID*
  - *SID*, *CID* → *grade*
- { *CID*, *email* }$^{+}$ = ?
- Closure = {*CID, email*}
- *email* → *SID*
  - Add *SID*; closure is now { *CID*, *email*, *SID* }
- *SID* → *name*, *email*
  - Add *name*, *email*; closure is now { *CID*, *email*, *SID*, *name* }
- *SID*, *CID* → *grade*
  - Add *grade*; closure is now all the attributes in *StudentGrade*

Using attribute closure

Given a relation R and set of FD's F

- Does another FD $X \rightarrow Y$ follow from F?
  - Compute X^+ with respect to F
  - If $Y \subseteq X^+$, then $X \rightarrow Y$ follow from F
- Is K a key of R?
  - Compute K^+ with respect to F
  - If K^+ contains all the attributes of R, K is a super key
  - Still need to verify that K is *minimal* (how?)

Data Normalization

- The process of validating and improving a logical design so that it satisfies certain constraints (*Normal Forms*) that avoid unnecessary duplication of data
  - Idea: decompose relations with anomalies to produce smaller, *well-structured* relations

- Using the Mapping Rules from ER translation to relational schema, we already get very close to a fully normalised schema.
  - Chances are good that it is already in 3NF (at least 2NF)
  - But to be sure we have to check…

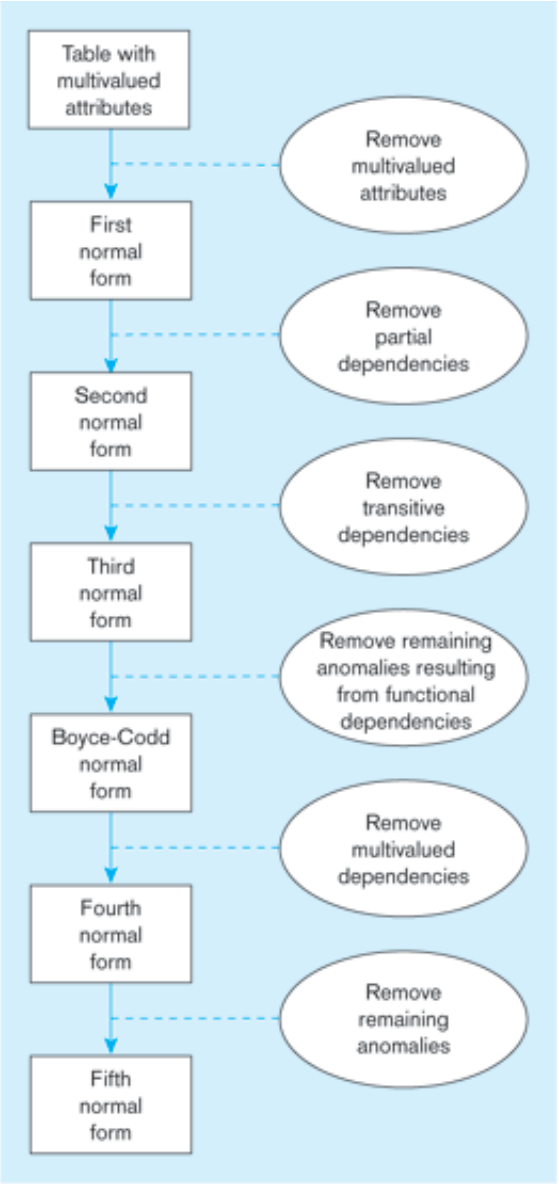
Table Decomposition

- Suppose that relation R contains attributes $A_1 \ldots A_n$.
  A *decomposition* of *R* consists of replacing *R* by two or more relations such that:
  - Each new relation scheme contains a subset of the attributes of *R* (and no attributes that do not appear in *R*), and
  - Every attribute of *R* appears as an attribute of one of the new relations.
  - All new relations differ.
- Central Idea of Normalisation:
  Decompose along a functional dependency.
  - **If *X -> Y* violates a normal form, decompose *R* into *R-Y* and *XY*.**
- Example:

  R (*A, B, C, D*) with FDs: {*AB* → *D* and *B -> C*}

Normal Forms

- Database theory identifies several normal forms for relational schemas.
- Each normal form is characterized by a set of restrictions.
- For a relation to be in a normal form it must satisfy the restrictions associated with that form.



First and Second Normal Form

- Remember last week:
  A relation *R* is in **first normal form (1NF)** if the domains of all attributes of *R* are *atomic*.
- Domain is atomic if its elements are considered to be indivisible units
  - Examples of non-atomic domains:
    - multivalued attributes, composite attributes
  - Non-atomic values complicate storage and encourage redundant (repeated) storage of data
- **Second Normal Form (2NF)** more of history value…
  - 1NF + every non-key attribute is fully functionally dependent on the primary key
  - This means: **No partial dependencies**
    (no FD $X \rightarrow Y$ where X is a strict subset of some key)

Non-key FD's

- Consider a non-trivial FD $X \rightarrow Y$ where X is not a primary key
  - Since X is not a super key, there are some attributes (say Z) that are not functionally determined by X

X	Y	Z
a	b	c_1
a	b	c_2
…	…	…

That b is always associated with a is recorded by multiple rows: redundancy, update anomaly, deletion anomaly

Example of redundancy

- *StudentGrade* (*SID*, *name*, *email*, *CID*, *grade*)
- *SID* → *name*, *email*

SID	*name*	*email*	*CID*	*grade*
142	Bart	bart@gmail.com	CPS116	C
142	Bart	bart@gmail.com	CPS114	B
123	Milhouse	milhouse@gmail.com	CPS116	B+
857	Lisa	lisa@gmail.com	CPS116	A
857	Lisa	lisa@gmai.com	CPS130	A
456	Ralph	ralph@gmail.com	CPS114	C
…	…	…	…	…

Decomposition

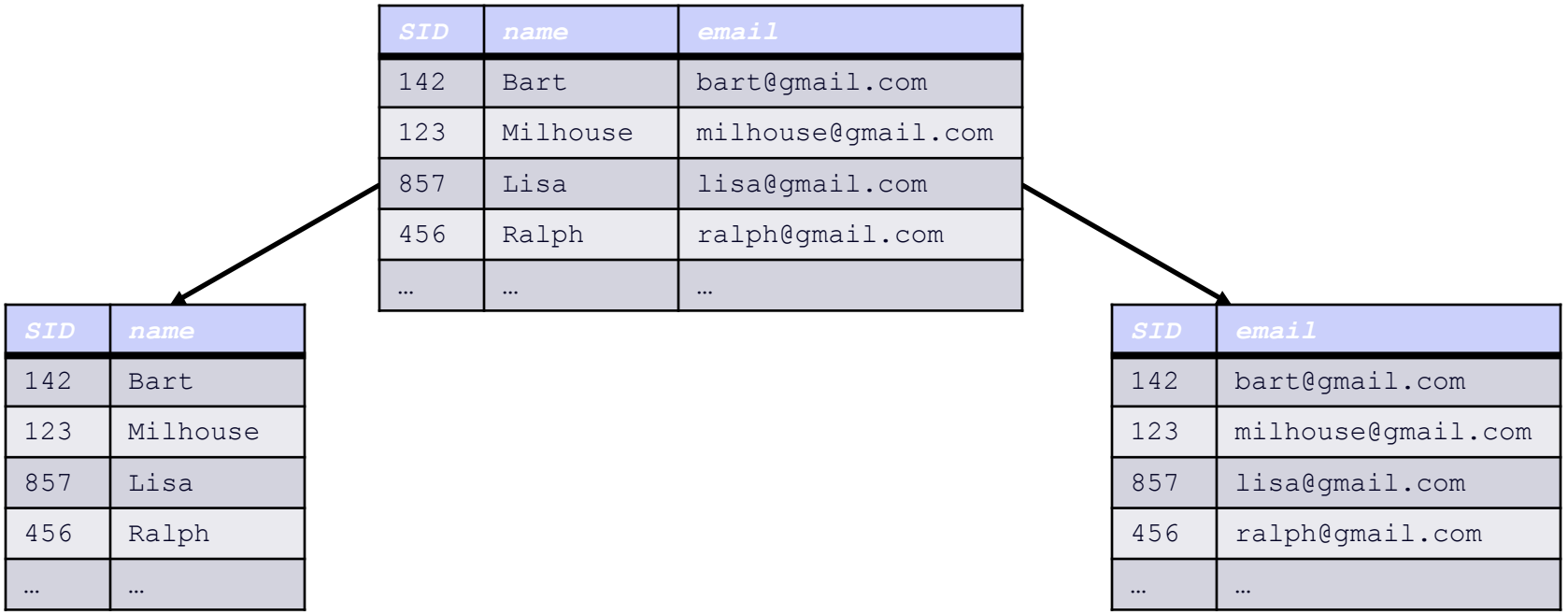
SID	name	email	CID	grade
…	…	…	…	…

SID	name	email
142	Bart	bart@gmail.com
123	Milhouse	milhouse@gmail.com
857	Lisa	lisa@gmail.com
456	Ralph	ralph@gmail.com
…	…	…

SID	CID	grade
142	CPS116	C
142	CPS114	B
123	CPS116	B+
857	CPS116	A
857	CPS130	A
456	CPS114	C
…	…	…

- Eliminates redundancy
- To get back to the original relation: $\bowtie$

Unnecessary decomposition



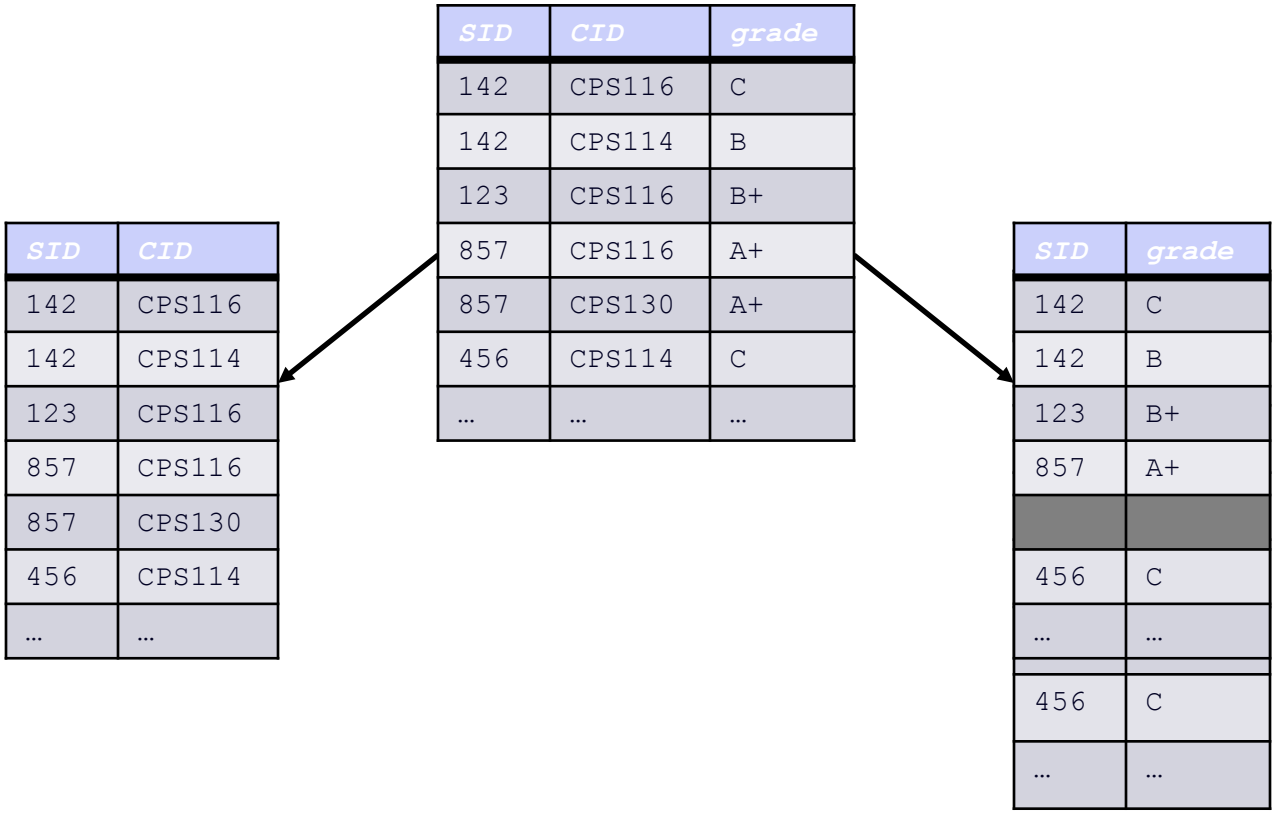
SID	name	email
142	Bart	bart@gmail.com
123	Milhouse	milhouse@gmail.com
857	Lisa	lisa@gmail.com
456	Ralph	ralph@gmail.com
…	…	…

SID	name
142	Bart
123	Milhouse
857	Lisa
456	Ralph
…	…

SID	email
142	bart@gmail.com
123	milhouse@gmail.com
857	lisa@gmail.com
456	ralph@gmail.com
…	…

- Fine: join returns the original relation
- Unnecessary: no redundancy is removed, and now *SID* is stored twice!

Bad decomposition



SID	CID	grade
142	CPS116	C
142	CPS114	B
123	CPS116	B+
857	CPS116	A+
857	CPS130	A+
456	CPS114	C
…	…	…

SID	CID
142	CPS116
142	CPS114
123	CPS116
857	CPS116
857	CPS130
456	CPS114
…	…

SID	grade
142	C
142	B
123	B+
857	A+
456	C
…	…
456	C
…	…

- Association between *CID* and *grade* is lost
- Join returns more rows than the original relation

Lossless join decomposition

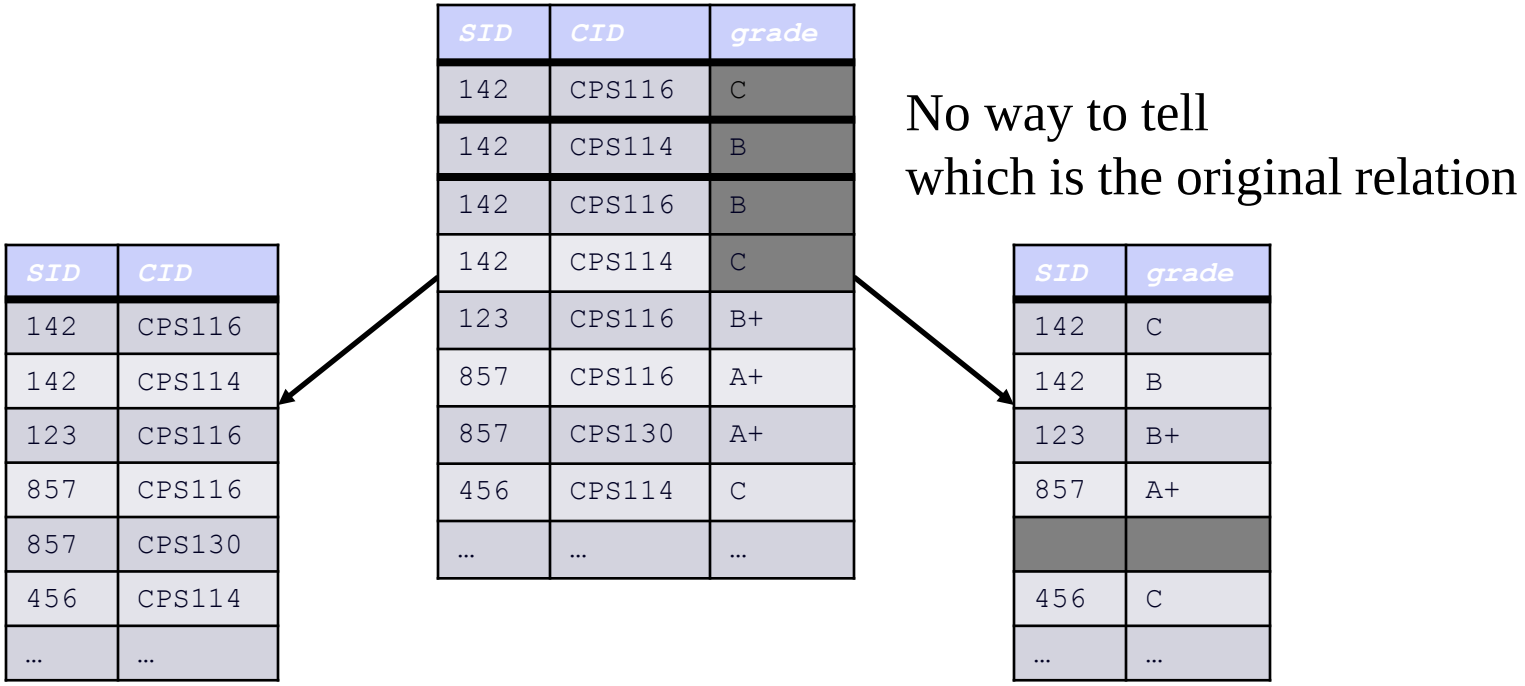
- Decompose relation *R* into relations *S* and *T*
  - *attrs*(*R*) = *attrs*(*S*) U *attrs*(*T*)
  - $S = \Pi_{attrs(S)} (R)$
  - $T = \Pi_{attrs(T)} (R)$
- The decomposition is a lossless join decomposition if, given known constraints such as FD's, we can guarantee that

$$R = S \bowtie T$$

- Any decomposition gives $R \subseteq S \bowtie T$ (why?)
  - A lossy decomposition is one with $R \subset S \bowtie T$

Loss? But I got more rows!

- "Loss" refers not to the loss of tuples, but to the loss of information
  - Or, the ability to distinguish different original relations



SID	CID	grade
142	CPS116	C
142	CPS114	B
142	CPS116	B
142	CPS114	C
123	CPS116	B+
857	CPS116	A+
857	CPS130	A+
456	CPS114	C
…	…	…

SID	CID
142	CPS116
142	CPS114
123	CPS116
857	CPS116
857	CPS130
456	CPS114
…	…

SID	grade
142	C
142	B
123	B+
857	A+
456	C
…	…

No way to tell
which is the original relation

Central Theorem of Schema Refinement

- Theorem

  A decomposition of R into R_1 and R_2 is lossless join if and only if at least one of the following dependencies is in F^+:

  - $R_1 \cap R_2 \rightarrow R_1$
  - $R_1 \cap R_2 \rightarrow R_2$

BCNF

- **A relation *R* is in Boyce-Codd Normal Form** if
  - For every non-trivial FD $X \rightarrow Y$ in R, X is a super key
  - That is, all FDs follow from "key $\rightarrow$ other attributes"

- **When to decompose**
  - As long as some relation is not in BCNF
  - ☞ **Then it is guaranteed to be a lossless join decomposition!**

BCNF decomposition algorithm

- Find a BCNF violation
  - That is, a non-trivial FD $X \rightarrow Y$ in R where X is not a super key of R
- Decompose R into R_1 and R_2, where
  - R_1 has attributes X U Y
  - R_2 has attributes X U Z, where Z contains all attributes of R that are in neither X nor Y
- Repeat until all relations are in BCNF

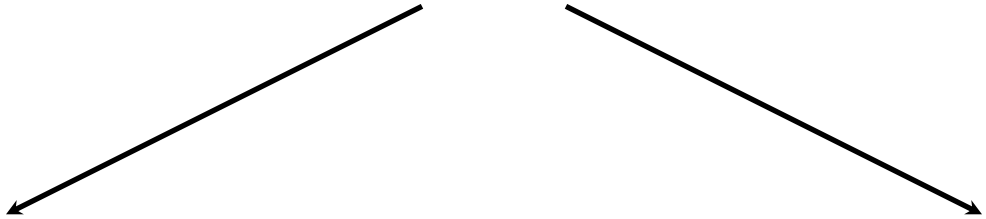
32

BCNF decomposition example

SID → *name*, *email*
email → *SID*
SID, *CID* → *grade*

StudentGrade (*SID*, *name*, *email*, *CID*, *grade*)

BCNF violation: *SID* → *name*, *email*



Student (*SID*, *name*, *email*)

BCNF

Grade (*SID*, *CID*, *grade*)

BCNF

33

Another Decomposition

SID → *name*, *email*
email → *SID*
SID, *CID* → *grade*

StudentGrade (*SID*, *name*, *email*, *CID*, *grade*)

BCNF violation: *email* → *SID*

StudentID (*email*, *SID*)
BCNF

StudentGrade' (*email*, *name*, *CID*,*grade*)

BCNF violation: *email* → *name*

StudentName (*email*, *name*)
BCNF

Grade (*email*, *CID*, *grade*)
BCNF

Why is BCNF decomposition lossless

- Given non-trivial $X \rightarrow Y$ in R where X is not a super key of R, need to prove:
- Anything we project always comes back in the join:

$$R \subseteq \Pi_{XY}(R) \bowtie \Pi_{XZ}(R)$$

  - Sure; and it doesn't depend on the FD
- Anything that comes back in the join must be in the original relation:

$$R \supseteq \Pi_{XY}(R) \bowtie \Pi_{XZ}(R)$$

  - Proof makes use of the fact that $X \rightarrow Y$

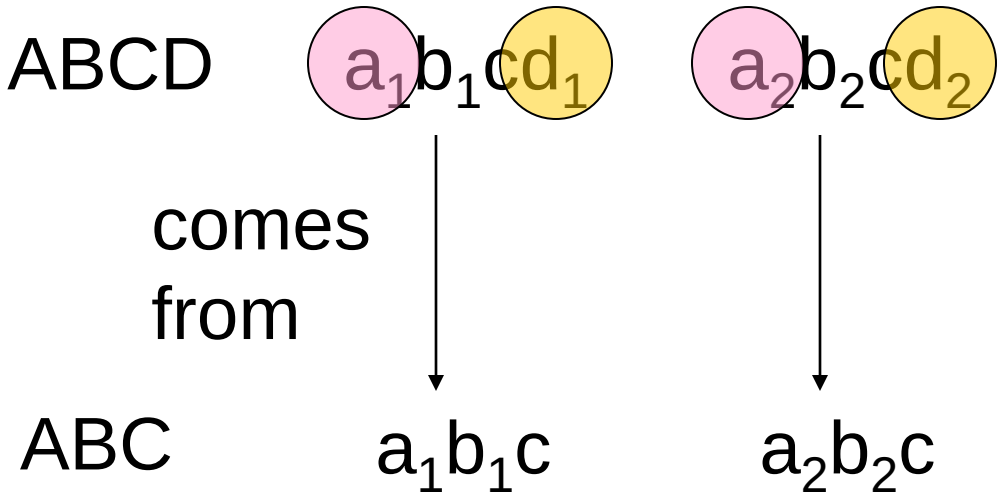
Finding Projected FD's

- Motivation: during normalization we break a relation schema into two or more schemas.

- Example: R(*ABCD)* with FD's
  - *AB ->C*, *C ->D*, and *D ->A*.
    - Decompose into R_1(*ABC*) and R_2 (*AD*) What FD's hold in R_1 ?
    - Not only *AB ->C*, but also *C ->A* !

36

Why?



$d_1=d_2$ because
C -> D

$a_1=a_2$ because
D -> A

Thus, tuples in the projection with equal C's have equal A's;
C -> A.

37

Basic Idea

1. Start with given FD's and find all *nontrivial* FD's that follow from the given FD's.
   - Nontrivial = right side not contained in the left.
2. Restrict to those FD's that involve only attributes of the projected schema.

38

Exponential Algorithm to compute Projected FDs

Given R and a set of FDs F that hold in R

Compute the set of FDs F_1 that hold in $R_1 = \Pi(R)$

1. For each set of attributes *X* in R_1 compute X^+ with respect to the original F.

1. Add $X \rightarrow A$ for all *A* in $X^+ - X$ to F_1

3. Finally, use only FD's involving projected attributes.

39

A Few Tricks

- No need to compute the closure of the empty set or of the set of all attributes.
- If we find X^+ = all attributes, so is the closure of any superset of X.

Dependency Preservation--Motivation

- There are some structures of FD's that cause trouble when we decompose.
- *AB ->C* and *C ->B*.
  - Example: *A* = street address, *B* = city, *C* = zip code.
- There are two keys, {*A*,*B* } and {*A*,*C* }.
- *C ->B* is a BCNF violation, so we must decompose into *AC*, *BC*.

41

We Cannot Enforce FD's

- The problem is that if we use *AC* and *BC* as our database schema, we cannot enforce the FD *AB ->C* by checking FD's in these decomposed relations.

- Example with *A* = street, *B* = city, and *C* = zip on the next slide.

42

An Unenforceable FD

street	zip
545 Tech Sq.	02138
545 Tech Sq.	02139

city	zip
Cambridge	02138
Cambridge	02139

Join tuples with equal zip codes.

street	city	zip
545 Tech Sq.	Cambridge	02138
545 Tech Sq.	Cambridge	02139

Although no FD's were violated in the decomposed relations, FD street city -> zip is violated by the database as a whole.

43

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 require computing joins, which is expensive.

Third Normal Form: Motivation

- There are some situations where
  - **BCNF is not dependency preserving, and**
  - **efficient checking for FD violation on updates is important**
- Solution: define a weaker normal form, called Third Normal Form (3NF)
  - Allows more redundancy compared to BCNF
  - But functional dependencies can be checked on individual relations without computing a join.
  - There is always a lossless-join, dependency-preserving decomposition into 3NF.

Third Normal Form (3NF)

- R is a relation schema, with the set F of FDs
- R is in 3NF if and only if
  - for each FD: $X \rightarrow \{A\}$ in F+
- Then
  - $A \in X$ (trivial FD), or
  - X is a superkey for R, or
  - A is prime attribute for R
- A prime attribute is an attribute that is part of a candidate key

3NF Example

- R = (B, C, E)
  F = {E→B, BC→E}
-

- Remember that you always have to find all candidate keys in order to determine the normal form of a table
- Two candidate keys: BC and EC
- E→B ; B is prime attribute
- BC→E ; BC is a candidate key
- None of the FDs violates the rules of the previous slide. Therefore, R is in 3NF

Redundancy in 3NF

•Bank-schema = (Branch B, Customer C, Employee E)
•F = {E→B, e.g., an employee works in a single branch
BC→E}, e.g., when a customer goes to a certain branch s/he is always served by the same employee

Branch	Customer	Employee
Evanston	Wong	Murphy
Evanston	Agrawal	Murphy
Wilmette	Wong	Jones
Wilmette	null	Ramirez

A 3NF table still has problems

- redundancy (e.g., we repeat that *Murphy* works at *Evanston branch*)
- need to use null values (e.g., to represent that Ramirez works at Wilmette even though he is not assigned any customers) –not permitted !

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: $\{A \rightarrow B, \quad B \rightarrow C\}$
  - Parts of a functional dependency may be redundant
    - E.g.: on RHS: $\{A \rightarrow B, \quad B \rightarrow C, \quad A \rightarrow CD\}$ can be simplified to $\{A \rightarrow B, \quad B \rightarrow C, \quad A \rightarrow D\}$
    - E.g.: on LHS: $\{A \rightarrow B, \quad B \rightarrow C, \quad AC \rightarrow D\}$ can be simplified to $\{A \rightarrow B, \quad B \rightarrow C, \quad A \rightarrow D\}$
- 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 $X \rightarrow Y$ in F.
  - Attribute A is extraneous in X if $A \in X$ and F logically implies $(F - \{X \rightarrow Y\}) \cup \{(X - A) \rightarrow Y\}$.
  - Attribute A is extraneous in Y if $A \in Y$ and the set of functional dependencies $(F - \{X \rightarrow Y\}) \cup \{X \rightarrow (Y - A)\}$ 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

Extraneous attributes (ctd)

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 $X \rightarrow Y$ in *F*.
- To test if attribute $A \in X$ is extraneous in X
  1. compute $(\{X\} - A)^+$ using the dependencies in *F*
  2. check that $(\{X\} - A)^+$ contains Y; if it does, *A* is extraneous in X

- To test if attribute $A \in Y$ is extraneous in Y
  1. compute X^+ using only the dependencies in
     $F' = (F - \{X \rightarrow Y\}) \cup \{X \rightarrow (Y - A)\}$,
  2. check that X^+ contains *A;* if it does, *A* is extraneous in Y

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*
      $X_1 \rightarrow Y_1$ and $X_1 \rightarrow Y_2$ with $X_1 \rightarrow Y_1 Y_2$
    Find a functional dependency $X \rightarrow Y$ with an
      extraneous attribute either in X or in Y
    If an extraneous attribute is found, delete it from $X \rightarrow Y$
  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: $A \rightarrow B$
  $B \rightarrow C$

Computing a Canonical Cover--example

- $R = (A, B, C)$
  $F = \{A \rightarrow BC$
  $B \rightarrow C$
  $A \rightarrow B$
  $AB \rightarrow C\}$
- **Combine $A \rightarrow BC$ and $A \rightarrow B$ into $A \rightarrow BC$**
  - Set is now $\{A \rightarrow BC, B \rightarrow C, AB \rightarrow C\}$
- **A is extraneous in $AB \rightarrow C$**
  - 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: $A \rightarrow B$
  $B \rightarrow C$

Algorithm for 3NF Synthesis

- Let R be the initial table with FDs F
- Compute the canonical cover F_C of F
- S=∅
- **for** each FD X→Y in the canonical cover F_C
  if none of the schemas in S contain XY
  then S=S∪(X,Y)
- if no scheme contains a candidate key for R
- Choose any candidate key CN
- S=S ∪ table with attributes of CN

- **Note: unlike the BCNF algorithm where we break the original relation, in 3NF we synthesize the tables using the FDs in the canonical cover**

3NF Decomposition: An Example

- Relation schema:

  cust_banker_branch = (*customer_id, employee_id, branch_name, type*)

- The functional dependencies for this relation schema are:
  1. *customer_id, employee_id* $\rightarrow$ *branch_name, type*
  2. *employee_id* $\rightarrow$ *branch_name*
  3. *customer_id, branch_name* $\rightarrow$ *employee_id*
- We first compute a canonical cover
  - *branch_name* is extraneous in the r.h.s. of the 1st dependency
  - No other attribute is extraneous, so we get F_C =

    customer_id, employee_id $\rightarrow$ *type*
    employee_id $\rightarrow$ *branch_name*
    customer_id, branch_name $\rightarrow$ *employee_id*

3NF Decompsition Example (Cont.)

- The **for** loop generates following 3NF schema:

  (*customer_id, employee_id, type*)

  (*employee_id, branch_name*)

  (*customer_id, branch_name, employee_id)*

  - Observe that (*customer_id, employee_id, type*) contains a candidate key of the original schema, so no further relation schema needs be added

- Minor extension of the 3NF decomposition algorithm: at end of for loop, detect and delete schemas, such as (*employee_id, branch_name*), which are subsets of other schemas
  - result will not depend on the order in which FDs are considered
- The resultant simplified 3NF schema is:

  (*customer_id, employee_id, type*)

  (*customer_id, branch_name, employee_id)*