BLG 317E
DATABASE SYSTEMS

WEEK 9

Functional Dependencies (FD)

Determinant and Dependent

- Suppose X and Y are attributes (fields) in a relation (table).
- The expression **X → Y** means:
  If the value of X is known, then the value of Y can be obtained in table.
- X is the determinant and Y is the dependent attribute.

  The value X determines the value of Y.

  The value Y depends on the value of X.

  That is, if two rows have the same X value, then their Y values are the same as well.

- **CUSTOMER table example:**
  id → name
  id → address
  id → (name, address)
  (name , address) → phone

Functional Dependencies

- An FD is a form of integrity constraint about attributes (fields).
- An attribute is functionally dependent if its value is determined by another attribute.
- If we know the value of one or several data items, then we can find the value of another.

- If A $\rightarrow$ B, then for the same value of A, there can only be one value of B.
- If A $\rightarrow$ B and B $\rightarrow$ A, then relationship is one-to-one.
- If (A, B) $\rightarrow$ C, then the following is NOT necessarily true.
  A $\rightarrow$ C and B $\rightarrow$ C

Functional Dependency Examples

The followings are about **EMPOYEE** and **PROJECT** entities.

- **Example1:**
  Social security number of employee determines employee name.
  SSN → NAME

- **Example2:**
  Project number determines project name and project location.
  PNUMBER → (PNAME, PLOCATION)

- **Example3:**
  Employee SSN and Project number determines the hours per week that the employee works on the project.
  (SSN, PNUMBER) → HOURS

Functional Dependency Examples

The followings are about **CUSTOMER** entity.

- **Example4:**
  id → (name , address)
  Then followings are true.
  id → name
  id → address

- **Example5:**
  (name , address) → phone
  The following is NOT true.
  name → phone
  (There can be customers whose names are same.)

- **Example6:**
  phone → address
  address → phone

FD Inference Rules

Given a set of FDs, we can infer additional FDs,
by applying the Inference Rules below. (X, Y, Z are attributes.)

Inference Rules (Axioms)	Description
Reflexive Rule	If Y is subset of X, then X → Y
Augmentation Rule	If X → Y, then XZ → YZ
Transitive Rule	If X → Y and Y → Z, then X → Z
Decomposition Rule	If X → YZ, then X → Y and X → Z
Union Rule	If X → Y and X → Z, then X → YZ

Transitive Dependency

- An attribute is transitively dependent, if its value is determined by another attribute which is not a key.
- If B $\longrightarrow$ C , and B is not a key, then this is a transitive dependency.
- If A $\longrightarrow$ B , and B $\longrightarrow$ C ,
  Then, A $\longrightarrow$ C
  So, C attribute is transitively dependent.

Example:

id $\longrightarrow$ (name , address)

(name , address) $\longrightarrow$ phone

Then , **id $\longrightarrow$ phone**

So, phone attribute is transitively dependent.

Partial vs. Full Dependency

- A partial dependency exists when a determinant is part of a composite key.
- A functional dependency X → Y is a partial dependency, if there is some attribute that can be removed from X (composite attribute) and yet the dependency still holds.
- The opposite of partial dependency is called full-functional-dependency.

- Example: If (A, B) → C, and B → C.
  The (A, B) is composite primary key.
  Then, B → C is a partial dependency, because only part of the primary key (B) is enough to determine the value of C.

  Example:

  (name , address) → phone

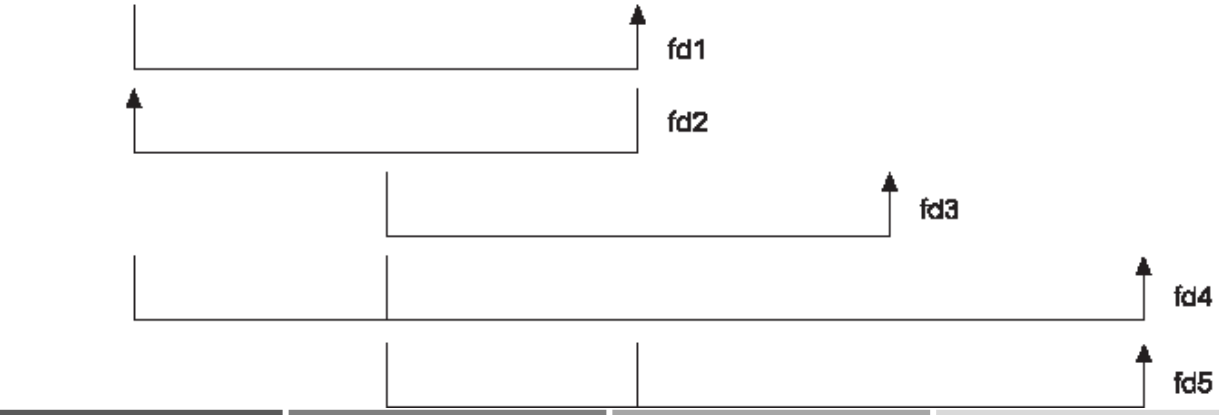
  Then , **address → phone** is a partial dependence

Example: Determining FDs from Sample Data

- Consider the data for attributes denoted A, B, C, D, and E in the sample relation (table) below.
- Assume that sample data values are representative of all possible values.
- To identify the functional dependencies that exist between attributes, we examine the table and identify when values in one column are consistent with the presence of particular values in other columns.

Sample Relation

A	B	C	D	E
a	b	z	w	q
e	b	r	w	p
a	d	z	w	t
e	d	r	w	q
a	f	z	s	t
e	f	r	s	t



Example: Determining FDs from Sample Data

- In result, the following functional dependencies between attributes are found.

A → C (fd1)

C → A (fd2)

B → D (fd3)

(A, B) → E (fd4)

(B, C) → E (fd5)

Minimal Set of FDs

Attribute Closure

- **Attribute Closure:**
  X = Attribute set
  Closure X+ = Set of all attributes B
  Such that : $X \longrightarrow B$

- It means X+ includes all attributes that are functionally determined from X.
- A closure represents a maximum set.

Example: Attribute Closure

Product (name, category, color, department, price)

- $name \longrightarrow color$
- $category \longrightarrow department$
- $color, category \longrightarrow price$

Attribute Closure:

- $\{name\}+ = \{name, color\}$
- $\{name, category\}+ = \{name, color, category, department, price\}$

Product table

name	category	color	department	price
Gizmo	Gadget	Green	Toys	40
Tweaker	Gadget	Brown	Toys	70
Gizmo	Stationery	Green	Office supplies	50

Uses of Attribute Closure

There are several uses of the attribute closure algorithm:

- Testing for superkey/candidate key:
  - To test if α is a superkey, we compute α^+, and check if α^+ contains all attributes of *R*.
    - if α is minimal, then it's a candidate key.
- 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 (see next!)
  - For each $\gamma \subseteq R$, we find the closure γ^+, and for each $S \subseteq \gamma^+$, we output a functional dependency $\gamma \rightarrow S$.

FD Closure

- **FD Closure:**
  - If F is a set of FDs, the closure F+ is the set of all FDs logically implied by F.
  - It represents the maximum set of FDs.

- F+ has the following properties:
  - **Soundness :** Any FD generated by an inference rule belongs in F+.
  - **Completeness:** Repeated application of the inference rules will generate all FDs in F+.

FD Closure Algorithm

- To compute the closure F+ of FDs, do the followings.

```
FOR EACH Subset of attributes X :
    Compute X+
    FOR EACH Subset of attributes Y ⊆ X+ :
        Output the FD  X ⟶ Y
```

Minimal Set of FDs (Canonical Cover / Irreducible Set)

- If we have a set of Functional Dependencies, we get the simplest and irreducible form of FDs after reducing them.
- This is called the Irreducible Set, as we can't reduce the set further.
- Minimal Set (Minimal Cover) is the opposite of Closure.
- Because a minimal set is reduced to minimum, whereas a Closure represents the maximum set.
- The followings are 3 steps to find the Minimal Set of FDs.

Step1) Split (decompose) the all right-hand side attributes of all FDs.

Step2) Remove all redundant FDs.

Step3) Find extraneous attributes on the left-hand-side and remove them.

Example1: Finding Minimal Set of FDs

- Suppose the following functional dependencies are given.
- R (A, B, C, D) is relation (table).
- A,B,C,D are attributes.
- **Find the Minimal Set of FDs.**

$$
\begin{aligned}
A &\rightarrow B \\
B &\rightarrow C \\
D &\rightarrow ABC \\
AC &\rightarrow D
\end{aligned}
$$

Solution

- **Step1 :** Split (decompose) all the right-hand side attributes of all FDs.

$A \rightarrow B$

$B \rightarrow C$

$D \rightarrow A$

$D \rightarrow B$

$D \rightarrow C$

$AC \rightarrow D$

Left-hand-sides cannot be split.

Therefore, $AC \rightarrow D$ cannot be split as:

$A \rightarrow D$

$C \rightarrow D$

Solution

- **Step2 :** Remove all redundant FDs.
  Redundant FD is if we can derive one FD from another FD.

A → B	Check the redundance of A→B. A→B is not redundant.
B → C	B→C is not redundant.
D → A	D→A is not redundant.
~~D → B~~	D→B is redundant, because of D→ A and A→B. D→B can be derived, which means D→B is redundant. So, we **remove D→B** from the FDs set.
~~D → C~~	D→C is redundant, because of D→A, A→B, and B→C, D→C can be derived, which means D→C is redundant. So, we **remove D→C** from the FDs set.
AC → D	AC→D is not redundant.

Solution

- **Step3 :** Find extraneous attributes on the left-hand-side and remove them.

$A \rightarrow B$

$B \rightarrow C$

$D \rightarrow A$

$D \rightarrow C$

$AC \rightarrow D$

We should only check $AC \rightarrow D$, because its left side has two attributes.

Based on $A \rightarrow B$ and $B \rightarrow C$, we have $A \rightarrow C$. Therefore, C is extraneous, and can be removed.

Solution

- The resulting Minimum Cover of FDs are below.

$$A \rightarrow B$$

$$B \rightarrow C$$

$$D \rightarrow AC$$

$$A \rightarrow D$$

Example2: Finding Minimal Set of FDs

- Suppose the following FDs are given.
- **Find the Minimal Set of FDs.**

$$A \longrightarrow B$$

$$A, B, C, D \longrightarrow E$$

$$E, F \longrightarrow G, H$$

$$A, C, D, F \longrightarrow E, G$$

Solution

- **Step1 :** Split (decompose) the right-hand-sides.

$$
\begin{aligned}
A &\longrightarrow B \\
A, B, C, D &\longrightarrow E \\
E, F &\longrightarrow G \\
E, F &\longrightarrow H \\
A, C, D, F &\longrightarrow E \\
A, C, D, F &\longrightarrow G
\end{aligned}
$$

Solution

- **Step2 :** Remove the redundant FDs.
  (They can be removed, since these FDs are logically implied by the remaining FDs.)

$$
\begin{aligned}
A &\longrightarrow B \\
A, B, C, D &\longrightarrow E \\
E, F &\longrightarrow G \\
E, F &\longrightarrow H \\
\cancel{A, C, D, F} &\cancel{\longrightarrow E} \\
\cancel{A, C, D, F} &\cancel{\longrightarrow G}
\end{aligned}
$$

Solution

- **Step3 :** Remove extraneous attributes from left-hand-sides. (B can be removed because of the first FD.)

$$
\begin{aligned}
A &\longrightarrow B \\
A, \cancel{B}, C, D &\longrightarrow E \\
E, F &\longrightarrow G \\
E, F &\longrightarrow H
\end{aligned}
$$

Solution

- Final result is the Minimum FD set.

$$
\begin{aligned}
A &\longrightarrow B \\
A, C, D &\longrightarrow E \\
E, F &\longrightarrow G \\
E, F &\longrightarrow H
\end{aligned}
$$

Database Normalization

Database Normalization

- **Normalization** is a process of evaluating and correcting table structures to minimize data redundancies.
- It is a process used in **Logical Design** of a database, where the tables are normalized (decomposed) properly, so that data can be added / deleted / updated efficiently.
- Normalization rules were defined first by Edgar F. Codd in 1970.
- It is based on using keys and FDs of relation schemas.
- Although there are many algorithms and some semi-automatic tools, database normalization is done mostly manually.

Goals of Normalization

- **Make the database more efficient, by eliminating data redundancy.** (Data redundancy is unneeded repetition of data.)

- **Prevent data modification anomalies:**
  - Insert anomaly
  - Update anomaly
  - Delete anomaly

- Results:
  - Data is accurate.
  - Disk storage space is reduced.
  - Queries on a database run as fast as possible (for updates/inserts – reads may not benefit from normalization for some cases).

Example Table (Unnormalized)

- Suppose the table below (UNF:Unnormalized Form) stores information about:
  Student id and name, department name of student.
  Course names that a student is taking.
- The followings are problems:
  - There is data redundancy, which causes modification anomalies.
    The same department names, and same course names are repeated.
  - A student can not take more than 3 courses.
  - Some students take less than 3 courses, causing null course fields.

Table in UNF

stu_id	stu_name	dept_name	course1	course2	course3
1	Weiss	Economics	Business Intro	Programming1	
2	Austin	Computer Eng.	Programming1	Data Structures	Algorithms
3	Taylor	Computer Eng.	Compilers		
4	Tobias	Electronics Eng.	Circuit Design		
5	Rogers	Civil Eng.			

Repeating group of columns

Insertion Anomaly

- Insertion anomaly refers to a situation where one cannot insert a new tuple into a relation, because of a missing related data.
- For example, we cannot add a new department, unless at least one student is registered to it
  - (Management is not in the table since no one has registered yet, even though there is such a department).

Table in UNF

stu_id	stu_name	dept_name	course1	course2	course3
1	Weiss	Economics	Business Intro	Programming1	
2	Austin	Computer Eng.	Programming1	Data Structures	Algorithms
3	Taylor	Computer Eng.	Compilers		
4	Tobias	Electronics Eng.	Circuit Design		
5	Rogers	Civil Eng.			
??	**??**	**Management**	??	??	??

Update Anomaly

- An update anomaly refers to a situation where an update of a single data value requires multiple tuples (rows) of data to be updated.
- Example: Suppose the course name “Programming1” was changed to “Intro to Programming”.
- We would have to find all of the columns that could have this Course field and rename each one that was found.
- Ideally, we would only update the value once, in one location.

Table in UNF

stu_id	stu_name	dept_name	course1	course2	course3
1	Weiss	Economics	Business Intro	**Intro to Programming**	
2	Austin	Computer Eng.	**Intro to Programming**	Data Structures	Algorithms
3	Taylor	Computer Eng.	Compilers		
4	Tobias	Electronics Eng.	Circuit Design		
5	Rogers	Civil Eng.			

Deletion Anomaly

- Deletion anomaly refers to a situation where deletion of data about one particular entity, causes unintended loss of other data.
- Example: Suppose the student named "Tobias" quits the school, and his record needs to be deleted from the system.
- If we delete his row, we lose the record of the "Circuit Design" course, because it's not stored anywhere else. The same can be said for the "Electronics" department name.
- We should be able to delete one type of data, or one record without having impacts on other records we don't want to delete.

Table in UNF

stu_id	stu_name	dept_name	course1	course2	course3
1	Weiss	Economics	Business Intro	Programming1	
2	Austin	Computer Eng.	Programming1	Data Structures	Algorithms
3	Taylor	Computer Eng.	Compilers		
4	**Tobias**	**Electronics Eng.**	**Circuit Design**		
5	Rogers	Civil Eng.			

Normal Forms

- The followings are levels of normalized forms.
  (Higher levels are more stricter than lower levels.)

  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)

Normal Forms

Normal Form	Features
1NF	**There are no repeating groups.** (No multivalued attributes allowed.) Primary Key is identified.
2NF	**There are no partial dependencies.** All nonkey attributes are dependent on all parts of the keys.
3NF	**There are no transitive dependencies.**
BCNF	Every determinant is a candidate key.
4NF	There are no multivalued dependencies.

Normal Forms

- 1NF is just a Universal Relation Schema, and does not use FDs.
- 2NF, 3NF, BCNF are based on keys and FDs of a relation schema.
- 4NF is based on keys and multi-valued dependencies.
- 5NF is based on keys and join dependencies.

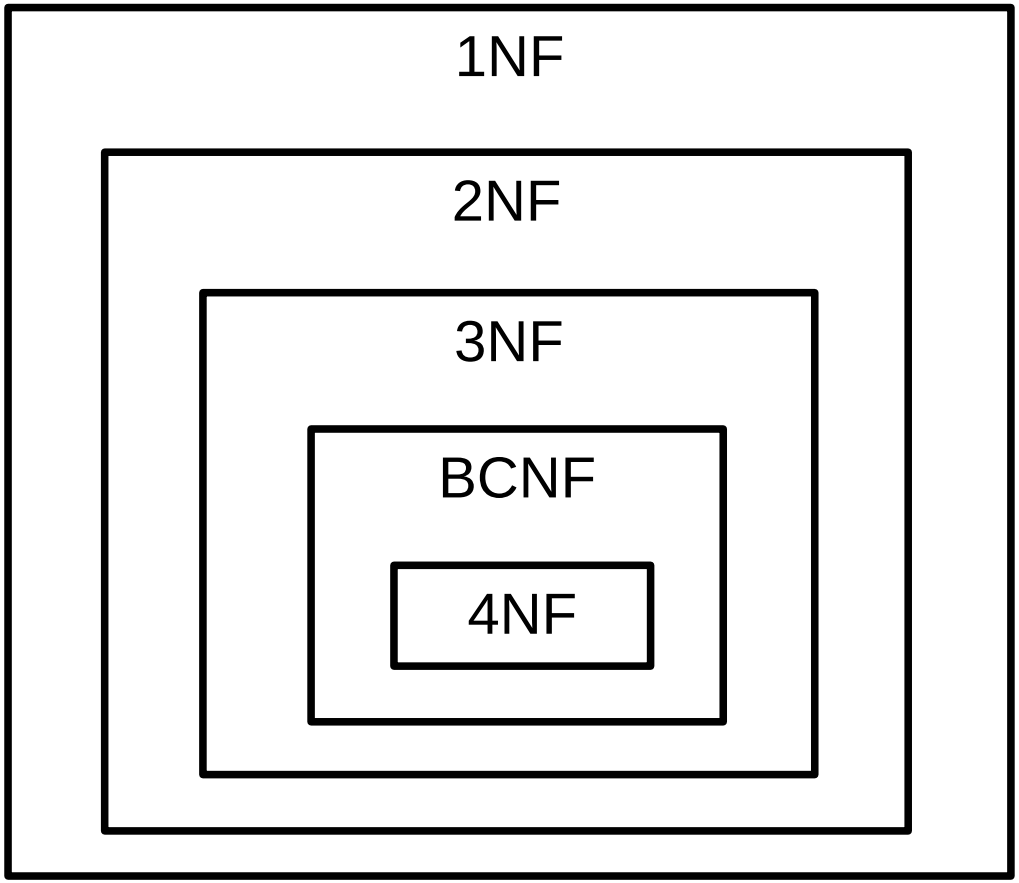
- **In practice, most of the databases are designed as 3NF (and BCNF if possible).**

Denormalization

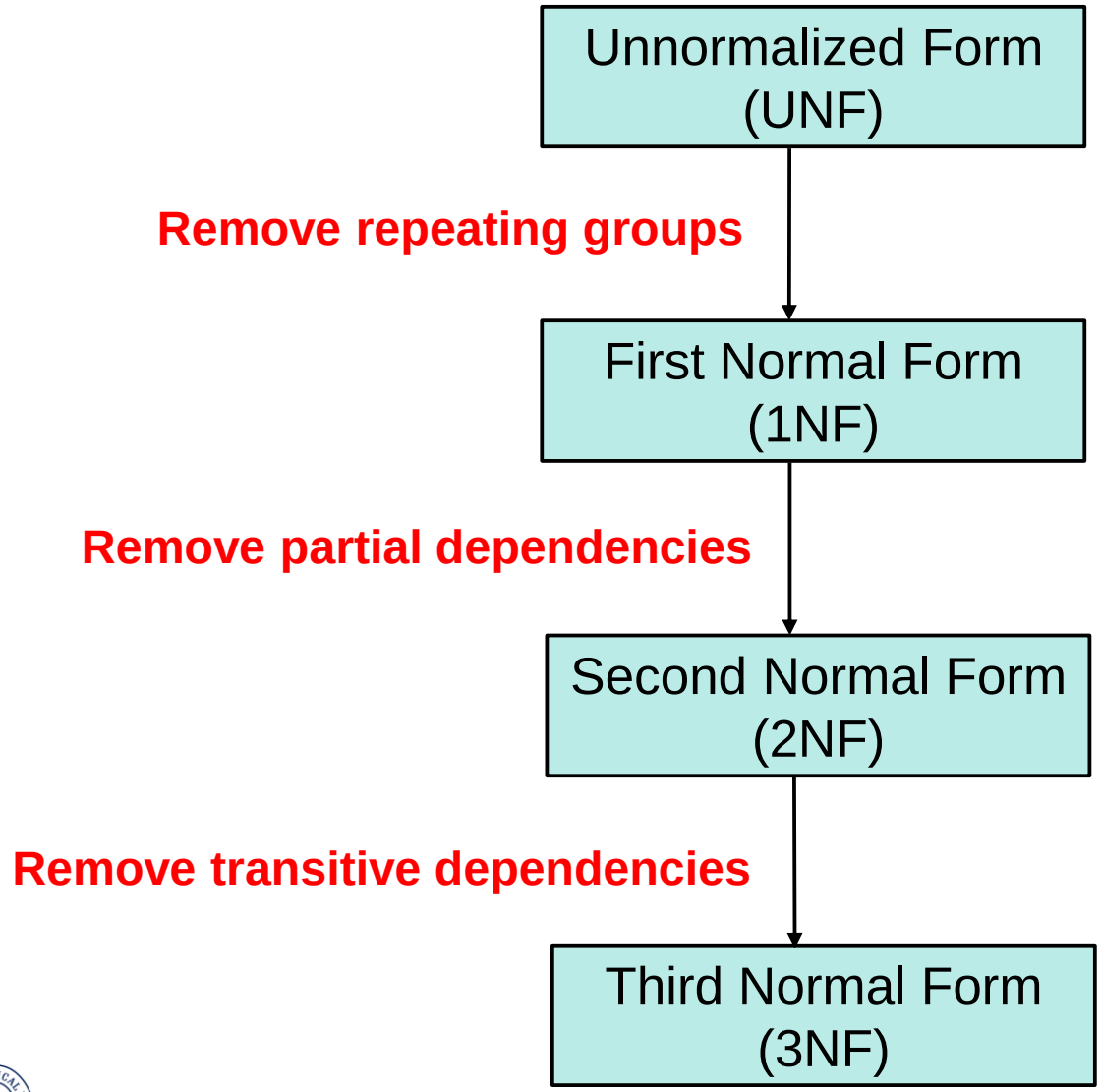
- Usually it is not necessary to normalize to the highest normal form.
- Higher level forms minimize data redundancy.
- But they may have too many tables, which requires many join operations when writing SQL queries, slowing the response time.
- Denormalization : When query speed is more important, a database can be re-organized to a lower form.
- For database applications which require only few data updates, but require many reads (select queries), then lower level normal forms will run faster.

Normal Forms

- Normal forms are incremental, higher level normal forms are already in lower levels.
- For example, if a relation is in 4NF, then it is also in BCNF, 3NF, 2NF, 1NF.



Normalization Steps



Normalization Process

- Start with a table called **Universal Relation Schema (1NF),** which contains all attributes in database.
- Specify all functional dependencies (FDs) among attributes, and construct a **minimal set of FDs**.
- Apply step-by-step **decompositions** on the universal relation schema, to construct a higher **normalized form**.

- **Universal Relation Schema:** A relation schema R = (A1, A2, …, An) that includes all the attributes. Every attribute name is unique.

- **Minimal Set of FDs:** A set of FDs is minimal if it satisfies the following conditions. It is also called as Canonical Set / Irreducible Set.
  - Every dependency in FD set has a single attribute at its right-hand-side.
  - We cannot remove any dependency from the FD set.

Schema Decomposition

- **Schema Decomposition** is the process of grouping and dividing (partitioning / splitting) a universal relation schema R, into a set of multiple relation schemas R1, R2, …, Rn.
- It is a part of the normalization process.
- Functional dependencies are used in the decomposition process, to identify good ways to split tables.
- A decomposition must have the following properties.
  - Lossless-join : No information is lost.
  - Dependency preserving : All FDs are preserved.

Lossless-join Property of Decomposition

- Lossless-join decomposition is a term for information not being lost.
- That is, we can reconstruct the original table by combining information from the two new tables, by means of natural join.
- The word loss in lossless refers to **loss of information**, not to loss of tuples.
- For “loss of information”, a better term is “**addition of spurious / false information**”.
- Lossless-join property can be called as **Non-additive** join property of a decomposition.

Lossless-join Property of Decomposition

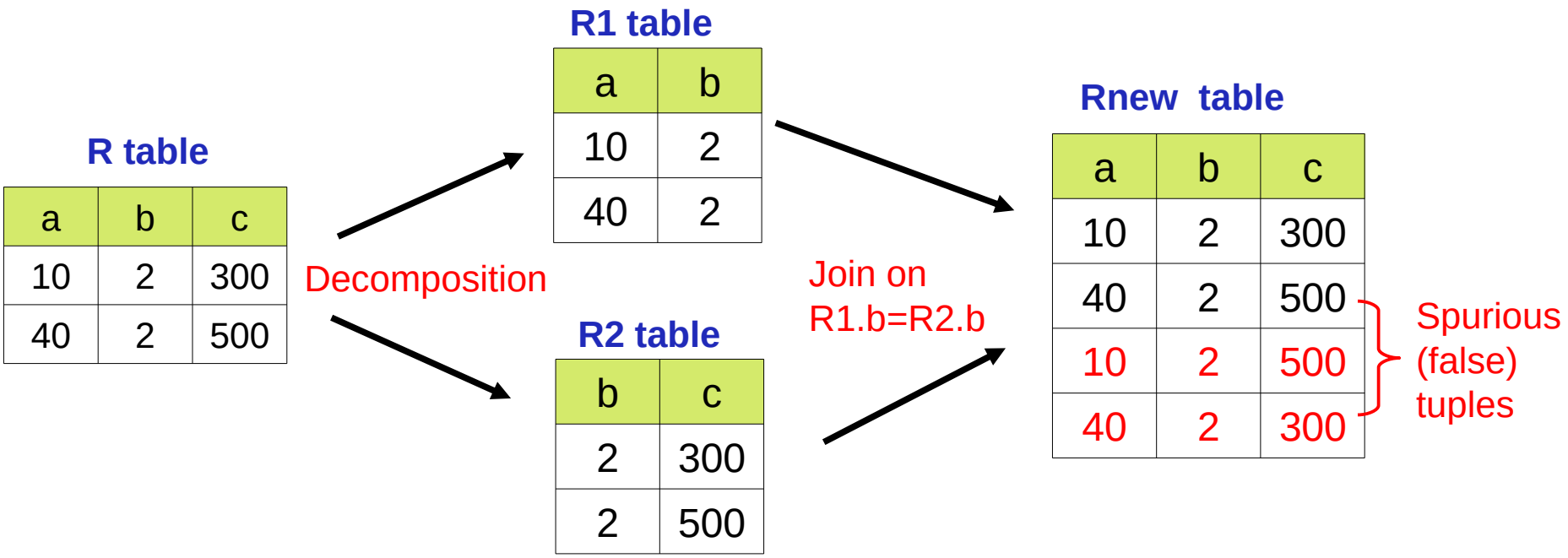
- Suppose relation R has been split into relations R1, R2, ..., Rn.
- The reconstruction of R from R1, ..., Rn
  can be done by means of the join operators.

- Join operation (natural join):
  - Concatenate those tuples of R1, R2, R3, ...
    which have same attribute name and same attribute value.
  - Eliminate the redundant attributes.

- If R = (R1 join R2 join R3 ...), then it is a lossles-join decomposition.
- Otherwise, it is a lossy-join decomposition.

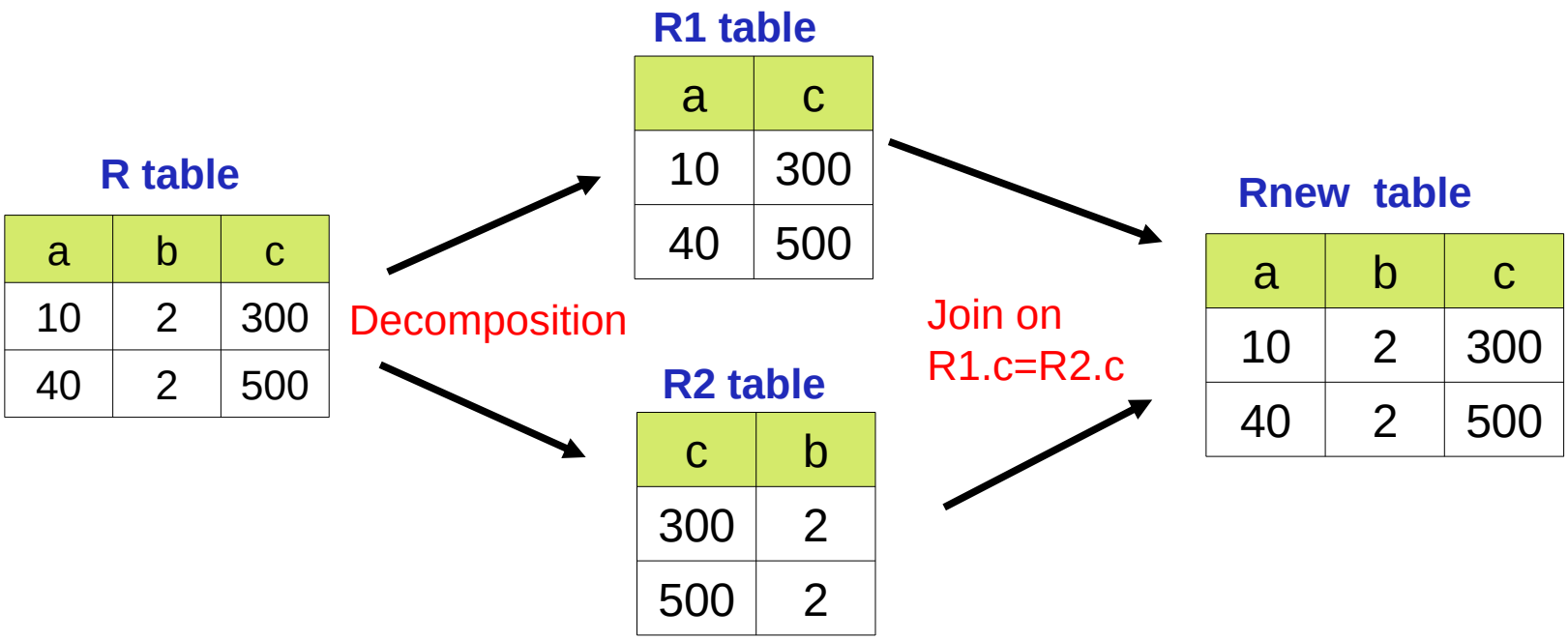
Example1: Lossy-join Decomposition

- The following is an example of lossy-join decomposition.
- Decomposition causes false-tuple-additions, when applying join later.
- After decomposition, a join operation is applied on R1 and R2.
- But the new table (Rnew) is not equal to the original table (R).
- Reason: The b column is not a key, it should not be used for decomposition.



Example2: Lossless-join Decomposition

- The following is a lossless-join decomposition.
- After decomposition, and applying a join operation later, the new table is equal to the original table.
- Reason: The c column is a key, it can be used for decomposition & join.



Normal Forms and Decomposition Rules

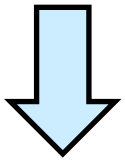
Normal Form	Requirements	Decomposition Rules
First	No multi-valued attributes	Form new relations for each multivalued attribute or repeating group
Second	Satisfy at least one of the following three conditions: Primary key consists of a single attribute No non-key attributes No non-key attribute should be functionally dependent on part of the primary key (every non-key attribute should be fully functionally dependent on the primary key)	Decompose and setup a new relation for each partial key with its dependent attribute(s). Make sure to keep a relation with the original primary key and any attributes that are fully functionally dependent on it
Third	No transitive dependencies. Relation should be in second normal form and should not have a non-key attribute functionally determined by another non-key attribute (or a set of non-key attributes)	Decompose and set up a new relation that includes the nonkey attribute(s) that functionally determine(s) the other nonkey attributes
BCNF	Every determinant is a candidate key	Decompose and set up a new relation that includes the non-candidate key attribute(s) that functionally determine(s) the other nonkey attributes.

First Normal From

- A table is in the first normal form if each attribute (field / column) contains only an atomic value.
- Atomic means that each attribute contains one value, not multiple values.
- A table should not contain any repeating column groups of data.
- For 1NF, create a separate row for each of values in multi-valued columns with redundancy.

1NF Normalization

EMP_ID	EMP_NAME	EMP_PHONE
14	John	7272826385, 9064738238
20	Harry	8574783832
12	Sam	7390372389, 8589830302



EMP_ID	EMP_NAME	EMP_PHONE
14	John	7272826385
14	John	9064738238
20	Harry	8574783832
12	Sam	7390372389
12	Sam	8589830302

Second Normal Form

- **Precondition**: Table should be already in 1NF.
- A table is in 2NF if no part of a composite Primary Key determines non-key attributes of the relation.
- Usually used in tables with a multiple-field primary key (composite key).
- Each non-key field should relate to the entire primary key.
- There should be no partial dependencies in table.
- A partial dependency happens when one or more non-key attribute(s) depends on a subset of the table's primary key.
- If there are partial dependencies, move it outside into a new table (Decompose relations such that there are no partial dependencies.)
- Also, columns that are not dependent on the primary key should be moved out to a separate table

→ because these columns might contain duplicated data.

Example : Table in UNF

- We have a table representing orders in an online store.
- Each row represents an item on a particular order made by a customer.
- Primary key is {Order, Product}
- Fields in Table:
  Order, Product, Quantity, UnitPrice, Customer, Address

Functional Dependencies in Example

- Each order is for a single customer.
  Order → Customer
- Each customer has a single address.
  Customer → Address
- Each product has a single price.
  Product → UnitPrice
- Transitive dependency:
  Order → Customer and **Customer → Address**
  Then, **Order → Address**

Normalization to 2NF

- Second normal form means no partial dependencies on candidate keys.
- Violating FDs:
  {Order} → {Customer, Address}
  {Product} → {UnitPrice}

- To remove the first FD, we define tables

  {Order, Customer, Address} (R1)

  and

  {Order, Product, Quantity, UnitPrice} (R2)

Normalization to 2NF

- R1 is now in 2NF, but there is still a partial FD in R2

  {Product} → {UnitPrice}

- To remove this, we define tables

  {Product, UnitPrice} (R3)

  and

  {Order, Product, Quantity} (R4)

Third Normal Form

- Precondition: Table should be already in 2NF.
- Prime attribute: Any attribute that is part of a composite candidate key.
- A table is in 3NF if all its attributes are determined only by its candidate keys, and not by any non-prime attributes.
- Usually used in tables with a single-field primary key.
- There should be no transitive dependencies.
- A transitive functional dependency happens when a non-prime attribute is dependent on another non-prime attribute.
- If there are transitive dependencies, move it outside into a new table (Decompose relations such that there are no transitive dependencies.)

- $X \rightarrow A$ is a "*good FD" if X is a (super) key*
- Or, if each column in A is part of some key

Normalization to 3NF

- R has now been split into 3 relations:

  R1: {Order, Customer, Address},
  R3: {Product, UnitPrice},
  R4: {Order, Product, Quantity}

Order → Customer
Customer → Address
Product → UnitPrice

  - R3 and R4 are in 3NF
  - R1 has a transitive FD on its key

- Violating FD

  {Customer} → {Address}

Transitive dependency:

Order → Customer and
Customer → Address
Then, Order → Address

- We define two tables from R1

  {Order, Customer}
  {Customer, Address}

Results of Normalizations

- **1NF:**

  {Order, Product, Customer, Address, Quantity, UnitPrice}

- **2NF:**

  {Order, Customer, Address},
  {Product, UnitPrice},
  {Order, Product, Quantity}

- **3NF:**

  {Product, UnitPrice},
  {Order, Product, Quantity},
  {Order, Customer},
  {Customer, Address}

Boyce-Codd Normal Form

- Precondition: Table should be already in 3NF.
- A table is in BCNF, if a non-prime attribute cannot determine a prime attribute.
- Every determinant should be a candidate key.

> • $X \rightarrow A$ is a "*good FD" if X is a (super) key*

Results of Normalizations

- **1NF:**

  {Order, Product, Customer, Address, Quantity, UnitPrice}

- **2NF:**

  {Order, Customer, Address},
  {Product, UnitPrice},
  {Order, Product, Quantity}

- **3NF:**

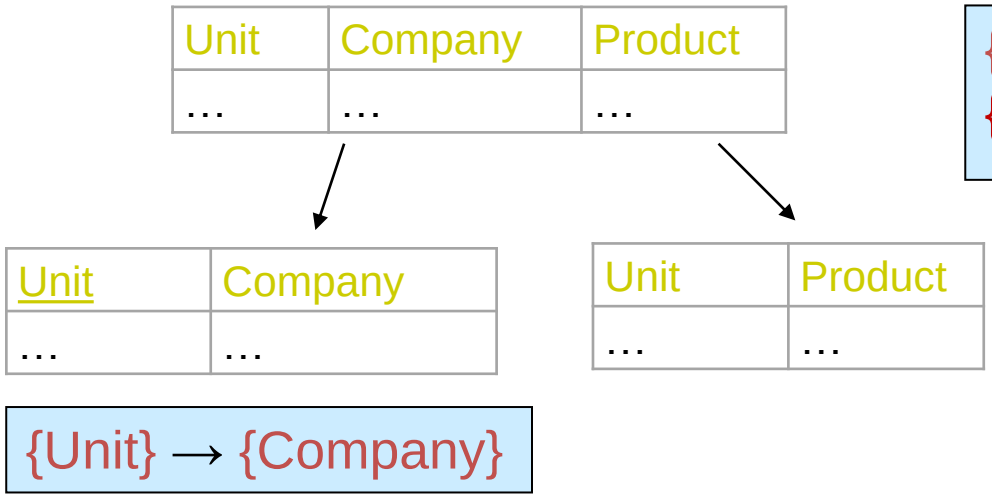
  {Product, UnitPrice},
  {Order, Product, Quantity},
  {Order, Customer},
  {Customer, Address}

- Each order is for a single customer.
  Order → Customer
- Each customer has a single address.
  Customer → Address
- Each product has a single price.
  Product → UnitPrice
- Transitive dependency:
  Order → Customer and
  Customer → Address
  Then, **Order → Address**

This design is also in BCNF

Dependency Preservation

- A problem with BCNF:



{Unit} → {Company}
{Company,Product} → {Unit}

We do a BCNF decomposition on a "bad" FD:
$\{Unit\}^+ = \{Unit, Company\}$

We lose the FD
{Company,Product} → {Unit}!!

When to prefer 3NF over BCNF

- There are some situations where
  - BCNF is not dependency preserving, and
  - Efficient checking for FD violation on updates is important
- Third Normal Form (3NF)
  - Allows some redundancy
  - But functional dependencies can be checked on individual relations without computing a join.
  - There is always a lossless-join, dependency-preserving decomposition into 3NF.

Redundancy in 3NF

- Consider the schema R below, which is in 3NF
  - $R = (J, K, L)$
  - $F = \{JK \rightarrow L, L \rightarrow K\}$
  - And an instance table:

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

- What is wrong with the table?
  - Repetition of information
  - Need to use null values (e.g., to represent the relationship l_2, k_2 where there is no corresponding value for J)

Design Goals

- Goal for a relational database design is:
  - BCNF.
  - Lossless join.
  - Dependency preservation.
- If we cannot achieve this, we accept one of
  - Lack of dependency preservation
  - Redundancy due to use of 3NF
- Interestingly, SQL does not provide a direct way of specifying functional dependencies other than superkeys.

  Can specify FDs using assertions, but they are expensive to test, (and currently not supported by any of the widely used databases!)
- Even if we had a dependency preserving decomposition, using SQL, we would not be able to efficiently test a functional dependency whose left hand side is not a key.

Another Normalization Example

Checking for First Normal Form

Table does not have repeated group attributes. Thus, it is already in 1NF.

ACTIVITY (SID, Activity, Fee)
Key: SID
Sample Data

SID	Activity	Fee
100	Skiing	200
150	Swimming	50
175	Squash	50
200	Swimming	50

FDs?

SID → Activity, Fee

Activity → Fee

Checking for Second Normal Form

Key is not composite. All of a relation's non-key attributes are dependent on all of the key. It is already in 2NF.

FDs
SID → Activity, Fee
Activity → Fee

ACTIVITY (SID, Activity, Fee)
Key: SID
Sample Data

SID	Activity	Fee
100	Skiing	200
150	Swimming	50
175	Squash	50
200	Swimming	50

Checking for Third Normal Form

Check if there is any functional dependency where determinant is non-prime!

ACTIVITY (SID, Activity, Fee)
Key: SID
Sample Data

SID	Activity	Fee
100	Skiing	200
150	Swimming	50
175	Squash	50
200	Swimming	50

FDs

SID → Activity, Fee

Activity → Fee

Violating FDs?

Activity → Fee

The determinant is a non-prime attribute.

STU-ACT (SID, Activity)
Key: SID

SID	Activity
100	Skiing
150	Swimming
175	Squash
200	Swimming

ACT-COST (Activity, Fee)
Key: Activity

Activity	Fee
Skiing	200
Swimming	50
Squash	50

Checking for BCNF

Check if all determinants are candidate keys!

STU-ACT (SID, Activity)
Key: SID

SID	Activity
100	Skiing
150	Swimming
175	Squash
200	Swimming

ACT-COST (Activity, Fee)
Key: Activity

Activity	Fee
Skiing	200
Swimming	50
Squash	50

FDs
SID → Activity
Activity → Fee

Violating FDs?
None

This design is already in BCNF.

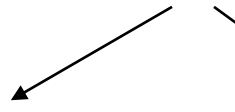
Example: Third Normal Form

HOUSING (SID, Building, Fee)
Key: SID
Functional dependencies:

- Building → Fee
- SID → Building, Fee

⇨ Violating FD

SID	Building	Fee
100	Randolph	1200
150	Ingersoll	1100
200	Randolph	1200
250	Pitkin	1100
300	Randolph	1200



STU-HOUSING (SID, Building)
Key: SID

SID	Building
100	Randolph
150	Ingersoll
200	Randolph
250	Pitkin
300	Randolph

BLDG-FEE (Building, Fee)
Key: Building

Building	Fee
Randolph	1200
Ingersoll	1100
Pitkin	1100

Example: Boyce -Normal Form

Every determinant attribute must be a candidate key.

ADVISER (SID, Major, Fname)

Key (primary): (SID, Major)
Key (candidate): (SID, Fname)

Functional dependencies:

FName → Major
SID, Major → Fname
SID, Fname → Major

⇨ Violating FD

SID	Major	Fname
100	Math	Cauchy
150	Psychology	Jung
200	Math	Riemann
250	Math	Cauchy
300	Psychology	Perls
300	Math	Riemann

STU-HOUSING (SID, Building)
Key: SID

SID	Building
100	Randolph
150	Ingersoll
200	Randolph
250	Pitkin
300	Randolph

BLDG-FEE (Building, Fee)
Key: Building

Building	Fee
Randolph	1200
Ingersoll	1100
Pitkin	1100

Multi-valued Dependency

- A multivalued dependency is a multi-tuple-generating dependency.
- There can be multiple values for a dependent attribute.
- Multivalued dependency is written by double arrows.
- Example: Each course may have multiple books. Any student registered for that course should be associated with all the textbooks of the course.

Course $\rightarrow\!\!\rightarrow$ StudentId
Course $\rightarrow\!\!\rightarrow$ Textbook

Student ID	Course	Textbook
1	Math	Algebra
1	Math	Calculus
2	Math	Algebra
2	Math	Calculus
2	Art	Art History
3	History	American History

Ename $\rightarrow\!\!\rightarrow$ Pname
Ename $\rightarrow\!\!\rightarrow$ Dname

Ename	Pname	Dname
Smith	X	John
Smith	Y	Anna
Smith	X	Anna
Smith	Y	John

MVD (Cont.)

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

  $$Y, Z, W$$

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

  $$< y_1, z_1, w_1 > \in r \text{ and } < y_1, z_2, w_2 > \in r$$

  then

  $$< y_1, z_1, w_2 > \in r \text{ and } < y_1, z_2, w_1 > \in r$$

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

  $Y \rightarrow\rightarrow Z$ if $Y \rightarrow\rightarrow W$

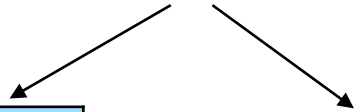
Fourth Normal Form

- A relation schema R is in **4NF** with respect to a set D of functional and multivalued dependencies if for all multivalued dependencies in D^+ of the form $\alpha \rightarrow\rightarrow \beta$, where $\alpha \subseteq R$ and $\beta \subseteq R$, at least one of the following hold:
  - $\alpha \rightarrow\rightarrow \beta$ is trivial (i.e., $\beta \subseteq \alpha$)
  - α is a superkey for schema R
- If a relation is in 4NF it is in BCNF

Decomposition into Fourth Normal Form

Course →→ StudentId
Course →→ Textbook

Student ID	Course	Textbook
1	Math	Algebra
1	Math	Calculus
2	Math	Algebra
2	Math	Calculus
2	Art	Art History
3	History	American History



Course	Textbook
Math	Algebra
Math	Calculus
Art	Art History
History	American History

Student ID	Course
1	Math
2	Math
2	Art
3	History