Relational Database Design(1)

Relational Database Design

- Introduction
- First Normal Form
- Pitfalls in Relational Database Design
- Functional Dependencies
- Decomposition
- Boyce-Codd Normal Form
- Third Normal Form
- Multivalued Dependencies and Fourth Normal Form
- Overall Database Design Process

Review of Basic Concepts

Terminology

- A relation is a table
- A relation schema is a table structure
  - Student (StdSSN, Fname, Lname)

Student

StdSSN	Fname	Lname

KEYs

- **Super key : a set of attributes having unique values**
- ***Candidate key : a minimum super key***
- ***Primary key : a designated candidate key,***
  - ***used for identifying a particular row.***
  - ***Identifying relationships among rows in different tables***

Examples of KEYs

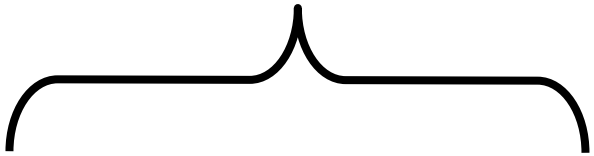
A table can have many keys.

- **Employee (EmpNum, Fname, Lname, CitizenID)**
  - **Super keys: EmpNum,** (EmpNum,Fname,Lname), CitizenID,…
  - **Candidate keys: EmpNum,** CitizenID
  - **Primary key: EmpNum**

Table Decomposition

- Need a primary key for referencing rows
- Avoid generation of spurious rows
  - When decomposing a table into 2 tables, be assure that joining them will get the same data as in the original one.
    - The same number of rows
    - The same values

Columns with NULL



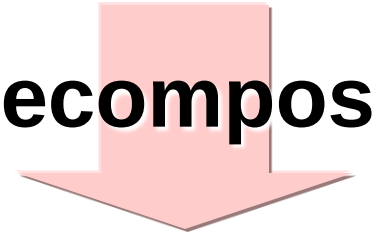
Employee

Name	Address	Province	Phone1	Phone2
John	1	01	-	-
John	2	01	021111111	022222222
Mary	3	02	-	025555555
Sam	4	02	-	-

Using non-unique attribute

Employee

Name	Address	Province	Phone1	Phone2
John	1	01	-	-
John	2	01	021111111	022222222
Mary	3	02	-	025555555
Sam	4	02	-	-



Split into 2 tables

Employee

Name	Address	Province
John	1	01
John	2	01
Mary	3	02
Sam	4	02

Name is not unique

Telephone

Name	Phone
John	021111111
Mary	025555555
John	022222222

The results of
Employee X Telephone

Name	Address	Province	PhoneNumber
John	1	01	021111111
John	1	01	022222222
John	2	01	021111111
John	2	01	022222222
Mary	3	02	025555555

Pitfalls in Relational Database Design

- Relational database design requires that we find a **"good" collection of relation schemas**.
- A bad design may lead to
  - **Repetition of data.**
  - **Inability to represent certain information.**

Database Design Goals

- Avoid ***redundant*** data
- Ensure that relationships among attributes are represented
- Facilitate the checking of updates for violation of *database integrity constraints*.

Repetition of data

Causes

- Deletion Anomalies
- Insertion Anomalies
- Update Anomalies

Example of *Redundant Data*

- Consider the relation schema:
  Lending-schema = (*branch-name, branch-city, assets, customer-name, loan-number, amount)*

branch-name	*branch-city*	*assets*	*customer-name*	*loan-number*	*amount*
Downtown	Brooklyn	9000000	Jones	L-17	1000
Redwood	Palo Alto	2100000	Smith	L-23	2000
Perryridge	Horseneck	1700000	Hayes	L-15	1500
Downtown	Brooklyn	9000000	Jackson	L-14	1500

- Redundancy
  - Wastes space
  - Complicates updating

Deletion of *Redundant Data*

branch-name	*branch-city*	*assets*	*customer-name*	*loan-number*	*amount*
Redwood	Palo Alto	2100000	Smith	L-23	2000
Perryridge	Horseneck	1700000	Hayes	L-15	1500

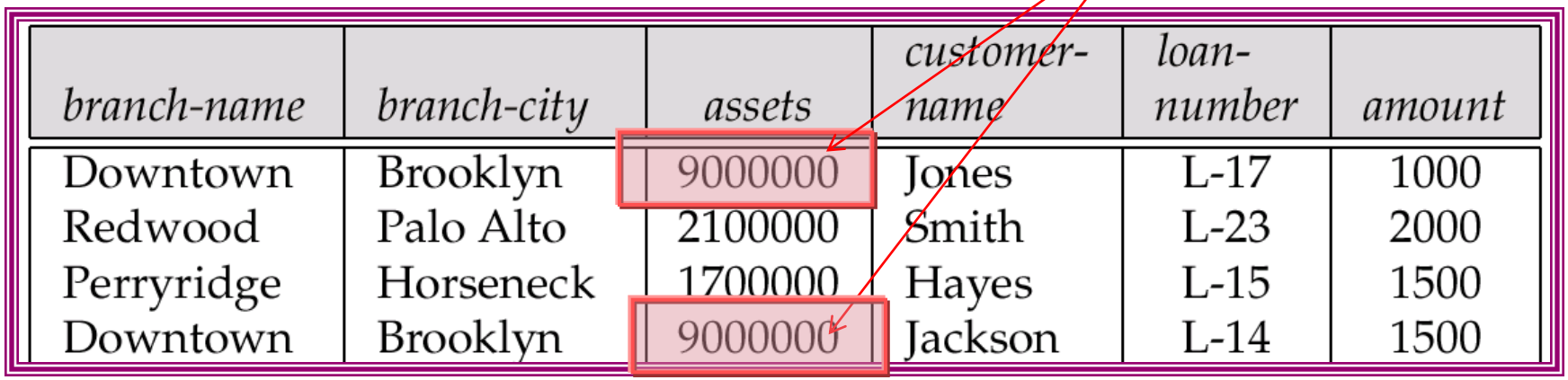
Deletion anomaly:
Loosing information of DOWNTOWN branch ?

Downtown	**Brooklyn**	**9000000**	–	–	–

- Can use null values, but they are difficult to handle.

Updating Anomaly

Update Downtown branch assets from 9000000 to 1 Million



branch-name	*branch-city*	*assets*	*customer-name*	*loan-number*	*amount*
Downtown	Brooklyn	9000000	Jones	L-17	1000
Redwood	Palo Alto	2100000	Smith	L-23	2000
Perryridge	Horseneck	1700000	Hayes	L-15	1500
Downtown	Brooklyn	9000000	Jackson	L-14	1500

Reducing the null values in rows

- **Avoid NULL values which may causes these problems:**
  - **Waste spaces**
  - **Confusing when using aggregated functions Count , Sum (*including NULL or not*)**

NULL Problem

EmpNo	Fname	Lname	Age
1	John	Smith	20
2	Jim	Jones	-
3	Janet	Green	40

Example (Oracle)

- **Count(age) return 2 ,**

  2 out of 3 supplied their age.

- **Count(*) return 3**

Data Quality Control

Domain Constraint

- **Constraint on attribute values**

  e.g., employee age between 20-60

- **Control by**
  - **DBMS**
  - **Applications**

Key Constraint

- **Key values must be unique.**
- **If a column is designated as a Primary Key of a table, DBMS will ensure the key constraint when inserting and updating the values.**

Entity Integrity Constraint

- **No primary key value can be null**
- **Unique value of a primary key**

Referential Integrity Constraint

- **Specified between two relations**
- **A row in one relation referring to another relation must refer to *an existing tuple***

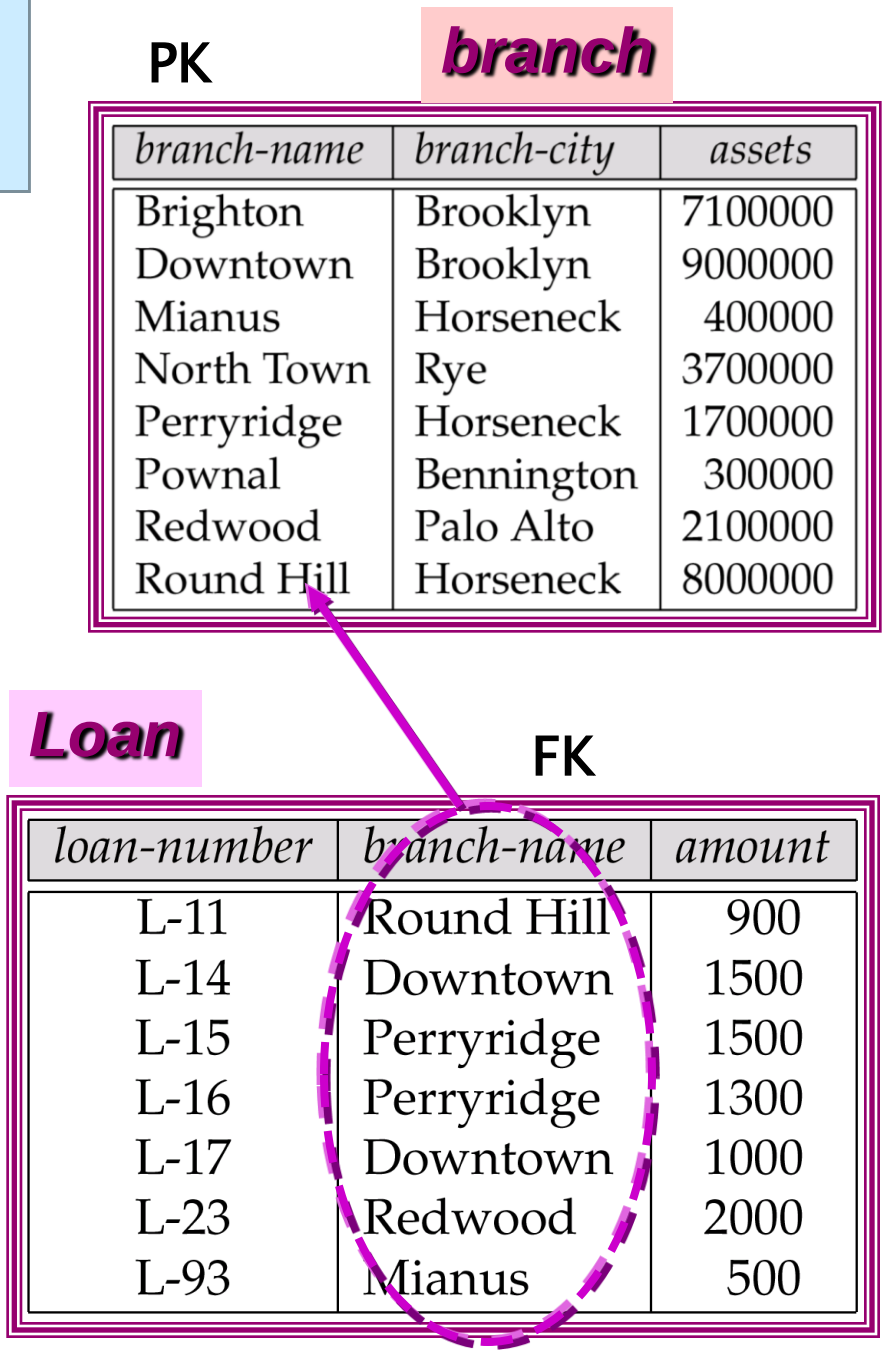
Ex. Loan Relation

Loan (*loan-number* ,*branch-name* , *amount*)

Foreign key

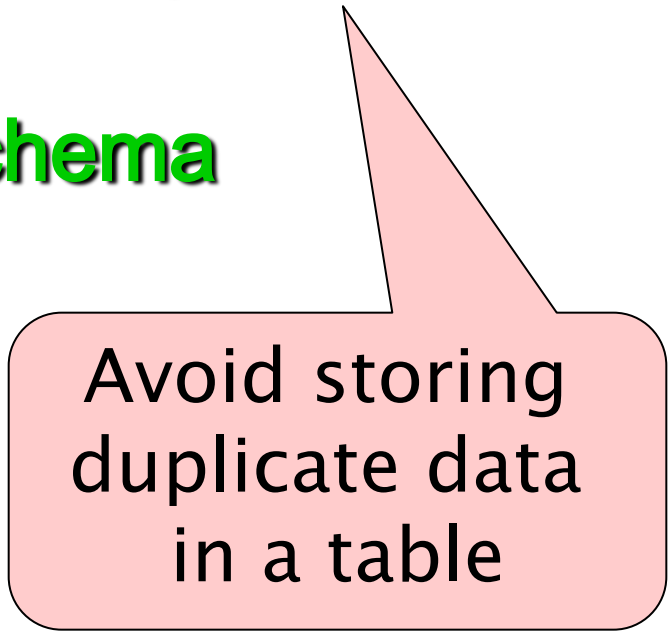
Referential Integrity Constraint (Cont.)

- Branch-name values must exist in the referenced table.
- Deleting a branch must ensure no referencing rows in Loan table.
- Can be checked by
  - DBMS
  - Application Program



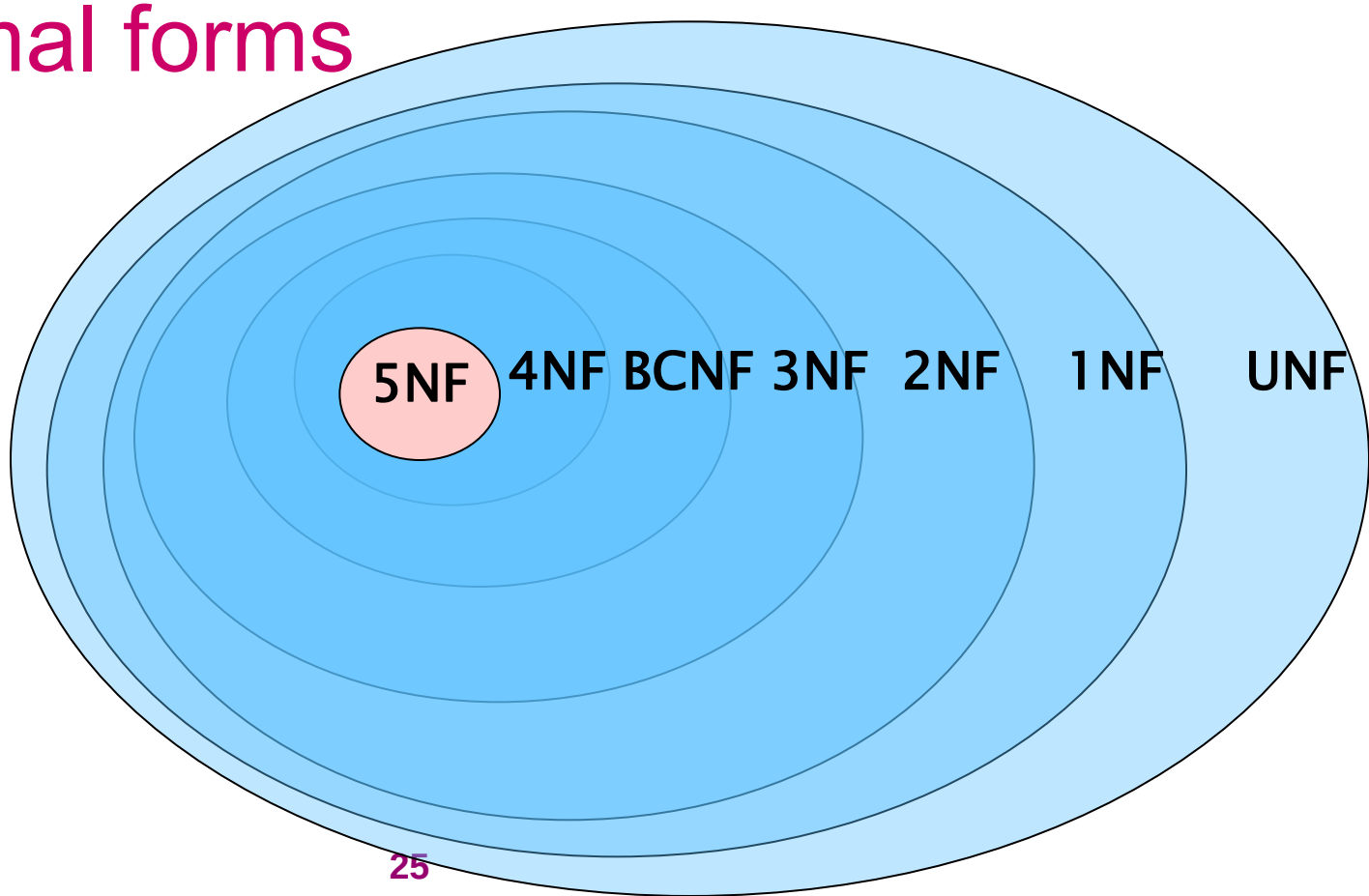
Normalization

- Check if a table schema is good ?
  - Yes, done
  - No, normalize that schema



How can we tell if a schema is good?

- No duplicate data
- Check the definitions of good forms
  - Normal forms



Schema Refinement

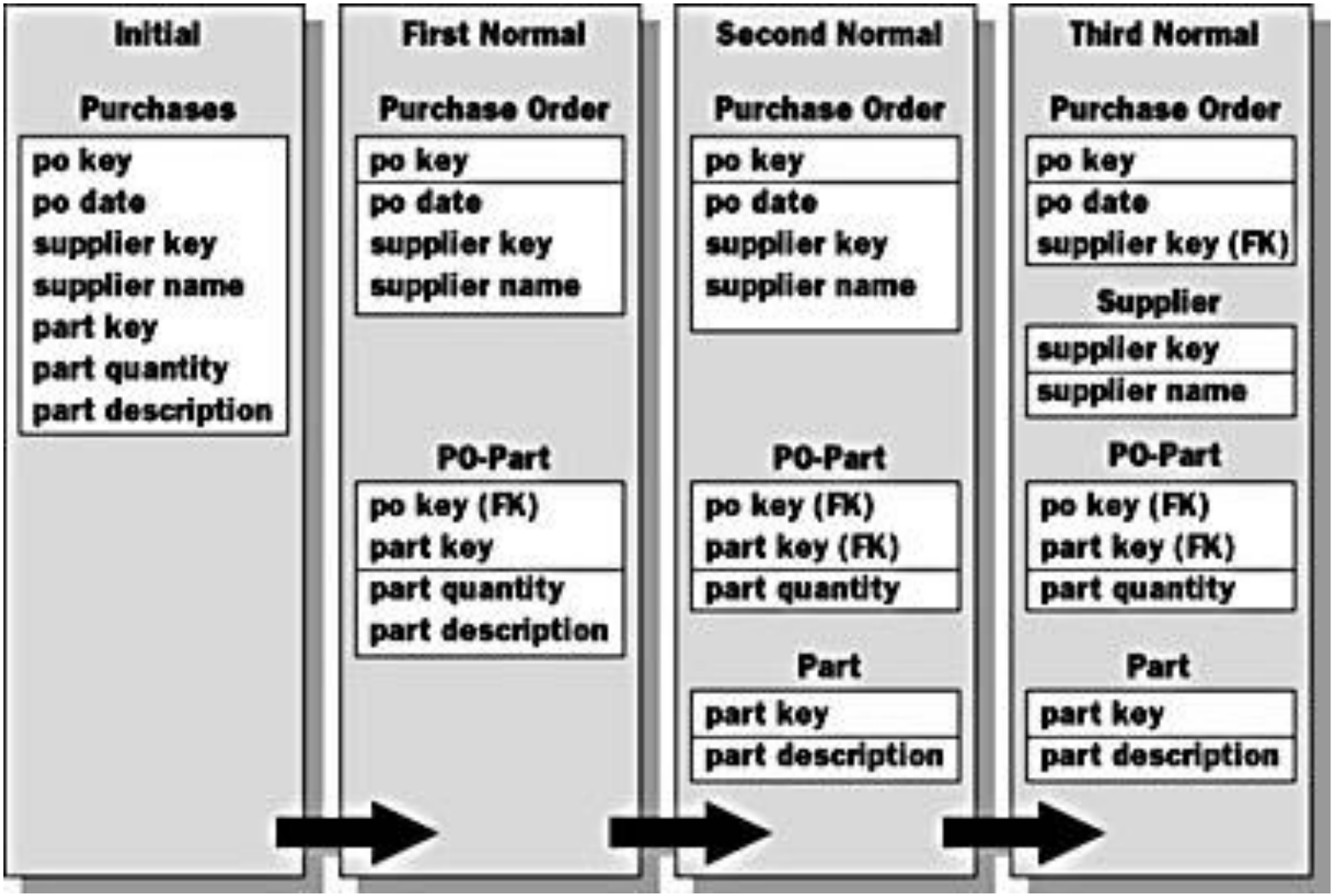
Normalization

- Eliminate
  - Deletion Anomalies
  - Insertion Anomalies
  - Update Anomalies

Avoid storing duplicate data in a table

By splitting the relation into two or more separated tables

Transforming a table into tables

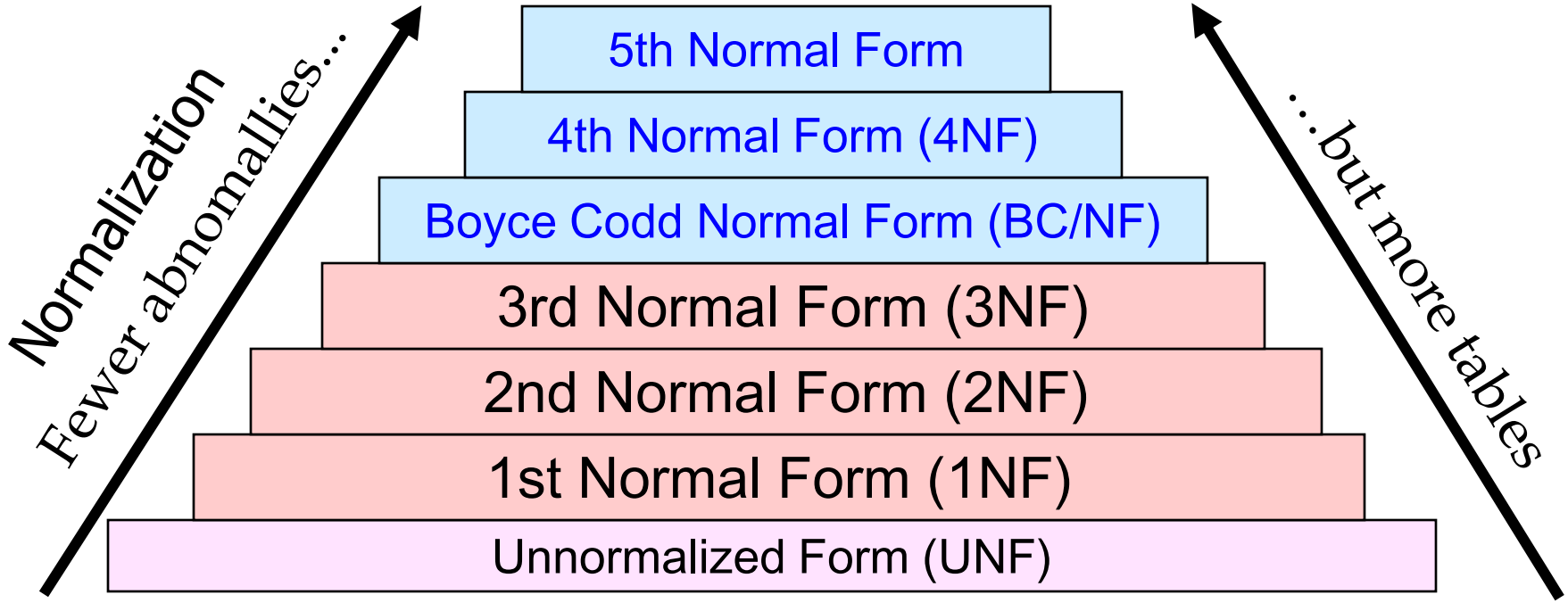


The Required Knowledge of Normalization

- Business rules
- Business processes
- Data relationships
- Normal form definitions

Normal Forms

- Normal forms are guidelines (steps) for the normalization process



29

Atomic value

- Domain is **atomic** if its elements are considered to be *indivisible units*
  - *E.g., integer, string, char*
  - **Examples of non-atomic domains:**
    - Set of names (Firstname Middle Lastname)
    - that stored in a column
    - Composite attributes: Course number

CS0012 or EE1127

department

First Normal Form

- A relational schema R is in the *first normal form* if the domains of all attributes of R are **atomic**

1NF

Employee (un-normalized)

emp_no	name	dept_no	dept_name	skills
1	Kevin Jacobs	201	R&D	C, Perl, Java
2	Barbara Jones	224	IT	Linux, Mac
3	Jake Rivera	201	R&D	DB2, Oracle, Java

Repeating groups

Employee (1NF)

emp_no	name	dept_no	dept_name	skills
1	Kevin Jacobs	201	R&D	C
1	Kevin Jacobs	201	R&D	Perl
1	Kevin Jacobs	201	R&D	Java
2	Barbara Jones	224	IT	Linux
2	Barbara Jones	224	IT	Mac
3	Jake Rivera	201	R&D	DB2
3	Jake Rivera	201	R&D	Oracle
3	Jake Rivera	201	R&D	Java

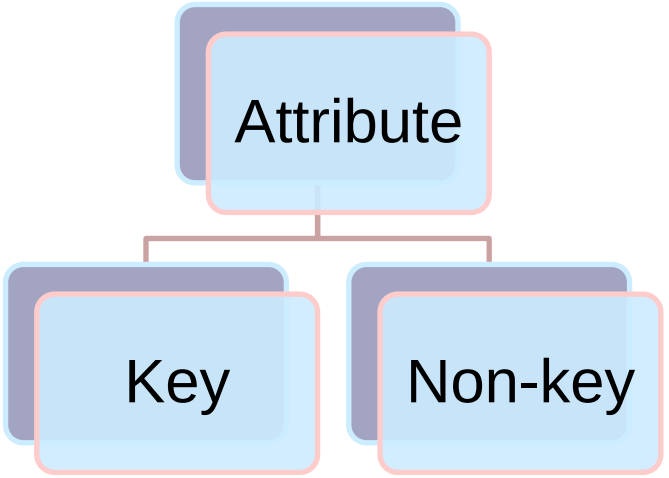
Key vs. Nonkey Attributes

Key Attribute :

attribute that is a part of the key or the key itself.

Non-Key Attribute :

attribute that is not a part of the key nor the key itself.



Key vs. Nonkey Attributes

A table can have many keys.

- **Employee (EmpNum, Fname, Lname, CitizenID)**

  KA NKA NKA KA

  - **Candidate keys:**
    - **EmpNum**
    - CitizenID
  - **Primary key: EmpNum**
  - **EmpNum is a key itself.**

Key vs. Nonkey Attributes

Enroll(*StudentID*, *CourseNo*, *Semester*, Grade**)**

KA KA KA NKA

(*StudentID*, *CourseNo*, *Semester*) is the primary key

e.g., StudentID is a part of a primary key.

Higher Normal Forms

- Normal forms higher than 1NF deal with *functional dependency (FD)*
- Identifying the normal form level by analyzing the *FD* between *attributes*
- *FD is a relationship among attributes*

36

Functional Dependencies

- If A and B are sets of attributes
- A → B
- A **uniquely determines** B

{StdSSN} → {FName}

{StdSSN, OfferNo} → {GPA}

{StdSSN} → {FName}

- Select StdSSN, Fname
- From Student
- Where StdSSN = '111'

StdSSN	Fname
111	John

For StdSSN 111, there exist only one corresponding first name.

Functional Dependencies

- Constraints on the set of legal relations.
- Require that the value for a certain set of attributes **determines uniquely** the value for another set of attributes.
- A functional dependency is a **generalization** of the notion of a ***key.***

Key determines $(A_1, A_2, .. A_n)$

FDs related to semantics of attributes

Functional Dependency Examples

- Dependency example
  - For each employee, there is only one corresponding department and salary, so:
    - EmpNum → Salary
    - EmpNum → Department

- Non-dependency example
  - Each instructor teaches multiple courses, so:
    - InstructorID does not determines CourseNumber

40

Functional Dependencies

R =(Province, Region)

R is a relation schema

r(R) is a table with schema R

table r

Province	Region
Bangkok	Central
Ayudthaya	Central
Phuket	South

Functional Dependencies (Cont.)

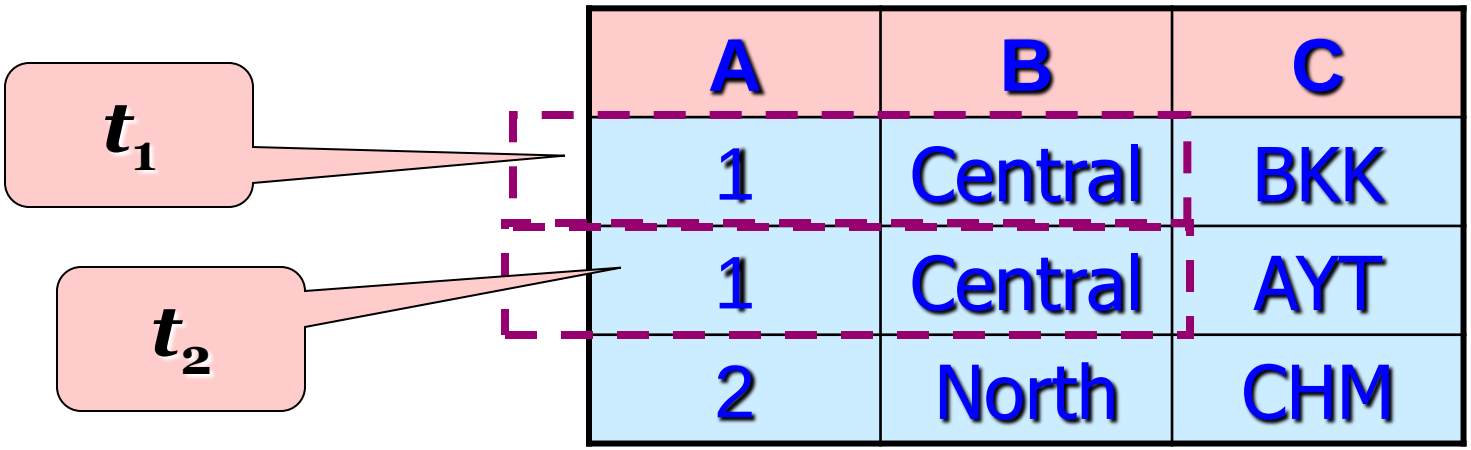
- Let R be a relation schema

$$A \subseteq R \quad and \quad B \subseteq R$$

- The functional dependency

$$A \rightarrow B \text{ holds on } R$$

if and only if for any legal relations $r(R)$,
if $t_1[A] = t_2[A]$ then $t_1[B] = t_2[B]$



Functional Dependencies (Cont.)

Example: Consider ***r(Province, Region)*** with the following instance of ***r.***

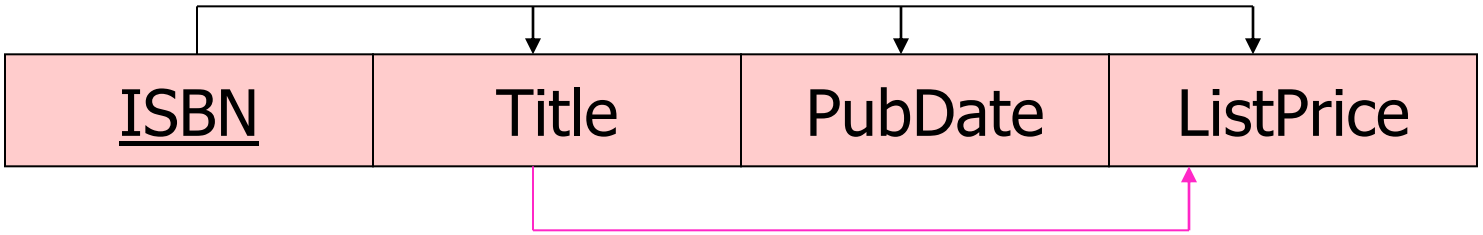
Province	Region
Bangkok	Central
Ayudthaya	Central
Phuket	South

- On this instance, Region $\rightarrow$ Province does **NOT** hold, Why not?
- but Province $\rightarrow$ Region does hold.

Functional Dependency and Keys

- By definition, a primary key (candidate key) functionally determines all other attributes

- Dependency diagram



1. **ISBN determines Title, PubDate and ListPrice**
2. **Title determines ListPrice**

Second Normal Form (2NF)

R is in 2NF if

- **R is in 1NF (*domains are atomic*) and**
- **Every nonkey attribute is dependent on *the whole key* *(not just a part of the key)*.**

  - It states the relationship between Non-Key attribute and the key.
  - Note: Key and key attribute are different

Second Normal Form (2NF)

Ex. Enroll(*StudentID, CourseNo*,
CourseName, Credit, Semester**)**

Key : (StudentID, CourseNo)

StudentID,CourseNo → CourseName, Credit ✓

CourseNo → CourseName, Credit ✗ **not 2NF****

A key

•dependent on a ***part of the key***.

Enroll

SID	***CNo***	CourseName	Credit	semester
101	111	Computer	3	2/46
102	111	Computer	3	2/46
103	111	Computer	3	2/46

repetition

Enroll*New*

SID	***CNo***	sem
101	111	2/46
102	111	2/46
103	111	2/46

Course*New*

CNo	Course Name	Credit
111	Computer	3
351	Database	3

Every table with a single attribute key is in Second Normal Form (2NF)

But 2NF still can cause anomalies.

since it states that every non-key attribute must depend on the whole key but *no restriction about*

- *dependency among non-key attributes or*
- *dependency among key attributes*

Third Normal Form (3NF)

It's in 2NF and has no transitive dependency. (X→Y, Y→Z, X→ Z)

Third Normal Form (3NF)

ID → (Type,Model) → Price

Equipment(ID ,Type , Model, Price) *in 2NF*

Key : ID

ID → Type , Model, Price ✓

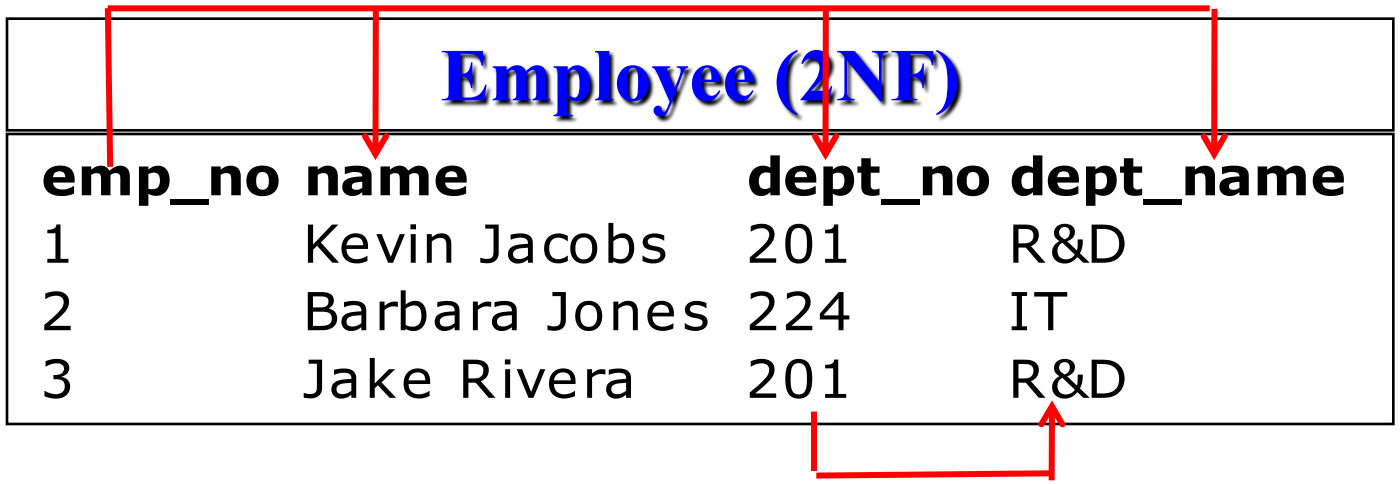
Type , Model → Price ✗ *transitive dependency*

split

Equipment (ID ,Type , Model)

EquipmentType (Type , Model, Price)

Transitive Dependence



emp_no -> name, dept_no, dept_name

dept_no -> dept_name (valid for 2NF but not 3NF)

There exists transitive FD.

However department can be considered a separate entity.

3NF

Employee (2NF)

emp_no	name	dept_no	dept_name
1	Kevin Jacobs	201	R&D
2	Barbara Jones	224	IT
3	Jake Rivera	201	R&D

split

Employee (3NF)

emp_no	name	dept_no
1	Kevin Jacobs	201
2	Barbara Jones	224
3	Jake Rivera	201

Department (3NF)

dept_no	dept_name
201	R&D
224	IT

Summary

3NF: If the tables are in 2NF, and every non-key attribute is dependent on the **key**, the **whole key**, and **nothing but the key**

Eliminate transitive dependencies

2NF: If the tables are in 1NF, and every non-key attribute is dependent on the **key**, the **whole key**

Eliminate partial dependencies

1NF: If the tables are relations

Split repeating groups in separate rows

UNF

Boyce-Codd Normal Form

- **R is in 3NF**
  1. *domains of all attributes of R are atomic*
  2. **all nonkey attributes are dependent on *the whole key*.**
  3. **no transitive dependency.**
- **and**
  4. **every determinant must be a superkey.**

Example (3NF but not BCNF):

SUPPLIER_PART

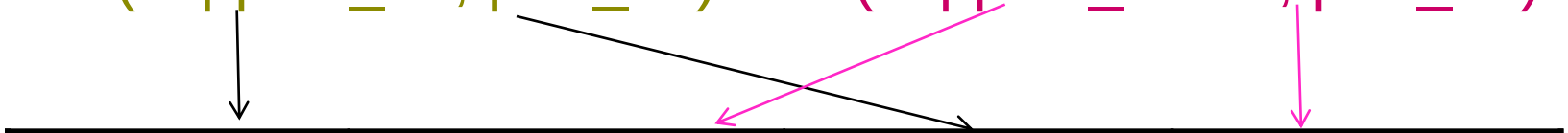
supplier_no	supplier_name	part_no	quantity
1	IBM	1	20
1	IBM	2	100
2	HP	1	50
2	HP	2	89
2	HP	3	70

X->Y, Y->X, where X->X is a reflexive not a transitive FD.

Example (3NF but not BCNF):

Functional Dependencies:

- assume that supplier_name's are always unique to each supplier. Thus we have two candidate keys:
  - (supplier_no, part_no) and (supplier_name, part_no)



supplier_no	supplier_name	part_no	quantity
1	IBM	1	20
1	IBM	2	100
2	HP	1	50
2	HP	2	89
2	HP	3	70

Example (3NF but not BCNF):

- Thus we have the following dependencies:
  - (supplier_no, part_no) -> quantity
  - (supplier_name, part_no) -> quantity
  - *supplier_name -> supplier_no*
  - *supplier_no -> supplier_name*
- No transtive dependency (in 3NF)
- **Not BCNF ,** determinant is not a superkey.
  - ***supplier_name*** *-> supplier_no*
  - ***supplier_no*** *-> supplier_name*

Decomposition (into BCNF):

SUPPLIER_PART

(supplier_no, supplier_name, part_no, quantity)

- Decompose the ***schema*** into:

SUPPLIER_ID (supplier_no, supplier_name)
SUPPLIER_PARTS (supplier_no, part_no, quantity)

Decompose 3NF into BCNF

SUPPLIER_PART (supplier_no, supplier_name, part_no, quantity)

SUPPLIER_ID (supplier_no, supplier_name)
SUPPLIER_PARTS (supplier_no, part_no, quantity)

supplier_no	supplier_name
1	IBM
2	HP

supplier_no	part_no	quantity
1	1	20
1	2	100
2	1	50
2	2	89
2	3	70

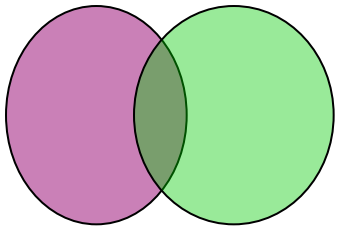
Goal — Devise a Theory for the Following

- **Decide** whether a particular relation ***R*** is in "good" form.
- In the case that a relation ***R*** is **not in "good" form, decompose** it into a set of relations $\{R_1, R_2, ..., R_n\}$ such that
  - each relation is in good form
  - the decomposition is a **lossless-join** decomposition
- Our theory is based on:
  - *functional dependencies*
  - *multivalued dependencies*

Decomposition

- All attributes of an original schema (*R*) must appear in the decomposition (R_1, R_2):

$$R = R_1 \cup R_2$$



- Lossless-join decomposition.
  For all possible relations *r* on schema *R*

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

Get the same result as r

Π = column projection
$\bowtie$ = cross product

Functional Dependencies (Cont.)

- *K* is a **superkey** for relation schema *R* if and only if $K \rightarrow R$
- *K* is a candidate key for *R* iff
  - $K \rightarrow R$, and
  - for no $Y \subset K$, $Y \rightarrow R$

minimal

e.g. R = (sid,name,grade)

sid,name → grade and sid→ name,grade

Hence {sid,name} is a super key.

but *{sid,name} is not* a candidate key.

Functional Dependencies (Cont.)

- Functional dependencies allow us to express constraints that cannot be expressed using superkeys.

Use of Functional Dependencies

- test relations legal under **F**.
  - *whether* r satisfies *F.*
- specify constraints on the set of legal relations
  - *F* holds on *R* if all legal relations on *R* satisfy *F.*

r1(R), r2(R) , r3(R)
tables r1, r2, r3 have the same schemas

Functional Dependencies (Cont.)

- **trivial** functional dependency

  E.g.

  - customer-name, loan-number → customer-name
  - customer-name → customer-name

  - In general, $Y \rightarrow X$ is **trivial** if $X \subseteq Y$

Closure of a Set of Functional Dependencies

- Given a set *F* set of functional dependencies, there are certain other functional dependencies that are *logically implied by* *F*.
  - E.g. If $A \rightarrow B$ and $B \rightarrow C$, then we can infer that $A \rightarrow C$
- The set of all functional dependencies logically implied by *F* is the *closure* of *F*.
- We denote the *closure* of *F* by F^+.

$$F = \{A \rightarrow B, B \rightarrow C\}$$
$$F^+ = \{A \rightarrow B, B \rightarrow C, A \rightarrow C\}$$

Closure of a Set of Functional Dependencies

- We can find all of F^+ by applying Armstrong's Axioms:
  - if $X \subseteq Y$, then $Y \rightarrow X$ (reflexivity)

    trivial rule: e.g. R = {A,B,C}
    $A \subseteq AB$ ดังนั้น $AB \rightarrow A$
  - if $Y \rightarrow X$, then $ZY \rightarrow ZX$ (augmentation)
  - if $Y \rightarrow X$, and $X \rightarrow Z$, then $Y \rightarrow Z$ (transitivity)
- These rules are
  - **sound** (generate only FDs that actually hold) and
  - **complete** (generate all FDs that hold).

Homework

- Consider 1NF, 2NF, 3NF, BCNF
- Compute F+ according to the Armstrong axioms

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^+
  - $A \rightarrow H$
    - $A \rightarrow B$ and $B \rightarrow H$ then $A \rightarrow H$ (transitivity)
  - $AG \rightarrow I$
    - $AG \rightarrow CG$ (by augmenting $A \rightarrow C$ with G)
    - $CG \rightarrow I$ (from F)
    - $AG \rightarrow I$ (transitivity)

Union Rule

$F = \{$ID→ name , ID→ dept $\}$

- ***ID → Name,Dept***
  - from ***ID→ Name and ID → Dept :*** "**union rule**" can be inferred from
    - FD definition,
    - ***ID → IDName*** by augmenting *ID → Name with ID*
    - ***ID Name→ Dept Name*** *by* augmenting *ID→Dept with Name,*
    - ***ID → Dept Name*** transitivity

$F = \{ A \rightarrow B, A \rightarrow C, CG \rightarrow H, C$

ID→ name
ID→ dept
ID→name,dept

- $CG \rightarrow HI$
  - from $CG \rightarrow H$ *and* $CG \rightarrow I$ *:* "**union rule**" can be inferred from
    - FD definition, **or**
    - $CG \rightarrow CGI$ by augmenting $CG \rightarrow I$ *with* CG
    - $CGI \rightarrow HI$ *by* augmenting $CG \rightarrow H$ *with* I,
    - $CG \rightarrow HI$ transitivity

It is
{C,G} → {H,I}

Procedure for Computing F^+

- To compute the closure of a set of FD F:

- $F^+ = F$

  repeat

  for each FD f in F^+

  apply reflexivity and augmentation rules on f

  add the resulting FD to F^+

  for each pair of FD f_1 and f_2 in F^+

  if f_1 and f_2 can be combined using transitivity
  then add the resulting FD to F^+

  until F^+ does not change any further

There exists an easier method

Closure of Functional Dependencies (Cont.)

- We can further simplify manual computation of F^+ by using the following additional rules.
  - **If** $Y \rightarrow X$ and $Y \rightarrow Z$ holds, **then** $Y \rightarrow XZ$ holds **(union)**
  - **If** $Y \rightarrow XZ$ holds, **then** $Y \rightarrow X$ and $Y \rightarrow Z$ holds **(decomposition)**
  - **If** $Y \rightarrow X$ and $ZX \rightarrow W$ holds, **then** $ZY \rightarrow W$ holds **(pseudotransitivity)**

  The above rules can be inferred from Armstrong's axioms.

Closure of Attribute Sets

- Given a set of attributes ***G*** , define the *closure* of ***G*** under *F* (denoted by G^+) as the set of attributes that are functionally determined by ***G*** under ***F***:

$$G \rightarrow X \text{ is in } F^+ \Leftrightarrow X \subseteq G^+$$

- Algorithm to compute G^+, the closure of ***G*** under ***F***

```
result := G;
while (changes to result) do
for each X → Z in F do
begin
        if X ⊆ result then  result := result ∪ Z
end
```

G^+ = Result

Example of Attribute Set Closure

- R = (A,B,C,G,H,I)
- F = { $A \rightarrow B$,
  $A \rightarrow C$,
  $CG \rightarrow H$,
  $CG \rightarrow I$,
  $B \rightarrow H$ }

- $(AG)^+$
  1. *result* = *A,G*
  2. *result* = *ABCG* ($A \rightarrow C$, $A \rightarrow B$ and $A \subseteq AG$)
  3. *result* = *ABCGH* ($CG \rightarrow H$ and $CG \subseteq ABCG$)
  4. *result* = *ABCGHI* ($CG \rightarrow I$ and $CG \subseteq ABCHG$)
- Is *AG* a candidate key?
  1. Is AG a super key?
     1. Does $AG \rightarrow R$? == Is $(AG)^+ \supseteq R$ yes
  2. Is any subset of AG a superkey?
     1. Does $A \rightarrow R$? == Is $(A)^+ \supseteq R$ no
     2. Does $G \rightarrow R$? == Is $(G)^+ \supseteq R$ no

Uses of Attribute Closure

- **Testing for superkey:**
  - compute Y^+, and check if $Y^+ \supseteq R$.
- **Testing functional dependencies**
  - To check if $Y \rightarrow X$ holds (or, is in F^+),
    - just check if $X \subseteq Y^+$.
  - a simple and cheap test, and very useful
- ***Computing closure of F***
  - For each $Z \subseteq R$, we find the closure Z^+, and
  - For each $S \subseteq Z^+$ ($Z^+ = \{ S, \ldots \}$)
    - we output a functional dependency $Z \rightarrow S$.

Computing closure of F

- R = (A,B,C,G,H)
- F = { A → B,
  B → C,
  CG → H }

For each $Z \subseteq R$, we find the closure Z^+,

$A^+ = \{A,B,C\}$
$B^+ = \{B,C\}$
$C^+ = \{C\}$
$G^+ = \{G\}$
$H^+ = \{H\}$
$CG^+ = \{C,G,H\}$
$AG^+ = \{A,B,C,G,H\} = R$
$BG^+ = \{B,C,G,H\}$

No need to compute AB^+, BC^+, AC^+ (trivial)

For each $S \subseteq Z^+$, output $Z \rightarrow S$.

- $F^+ = \{ A \rightarrow B, A \rightarrow C,$
  $CG \rightarrow H$, $B \rightarrow C$,
  $AG \rightarrow H$, $AG \rightarrow B$,
  $AG \rightarrow C$, $BG \rightarrow H$,
  $BG \rightarrow C \}$

Benefit of F^+

Normalization

Goals of Normalization

- Decide whether a particular relation *R* is in "good" form.
- In the case that a relation *R* is not in "good" form, decompose it into a set of relations $\{R_1, R_2, ..., R_n\}$ such that
  - each relation is in good form
  - the decomposition is a lossless-join decomposition
- Our theory is based on:
  - functional dependencies
  - multivalued dependencies

Decomposition

- All attributes of an original schema (*R*) must appear in the decomposition (R_1, R_2):

$$R = R_1 \cup R_2$$

- Lossless-join decomposition.
  For all possible relations *r* on schema *R*

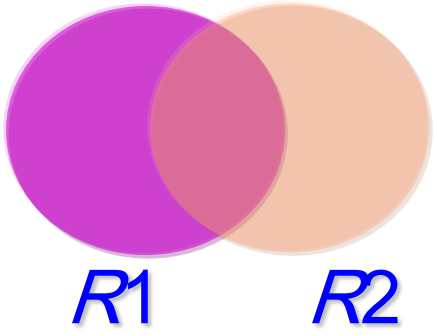
$$r = \Pi_{R1}(r) \bowtie \Pi_{R2}(r)$$

Result of join must = r

Decomposition

- 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^+:
  - $(R_1 \cap R_2) \rightarrow R_1$ **or**
  - $(R_1 \cap R_2) \rightarrow R_2$

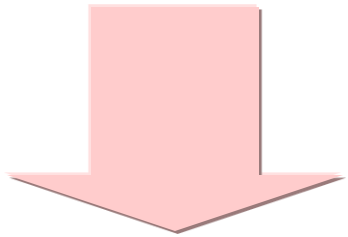
Key for joining R_1, R_2



Decomposition

Employee

Name	Address	Province	Phone1	Phone2
John	1	01	-	-
John	2	01	021111111	022222222
Mary	3	02	-	025555555
Sam	4	02	-	-



Decomposition

Employee(R1)

Name	Address	Province
John	1	01
John	2	01
Mary	3	02
Sam	4	02

R1 = { Name, Address, Province}
R2 = { Name, Phone }

R1 $\cap$ R2 = { Name }

Name $\twoheadrightarrow R_1$, Name $\twoheadrightarrow R_2$

Telephone(R2)

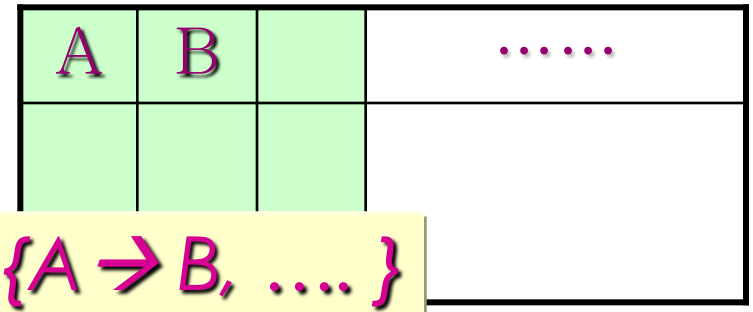
Name	Phone
John	021111111
Mary	025555555
John	022222222

Normalization Using Functional Dependencies

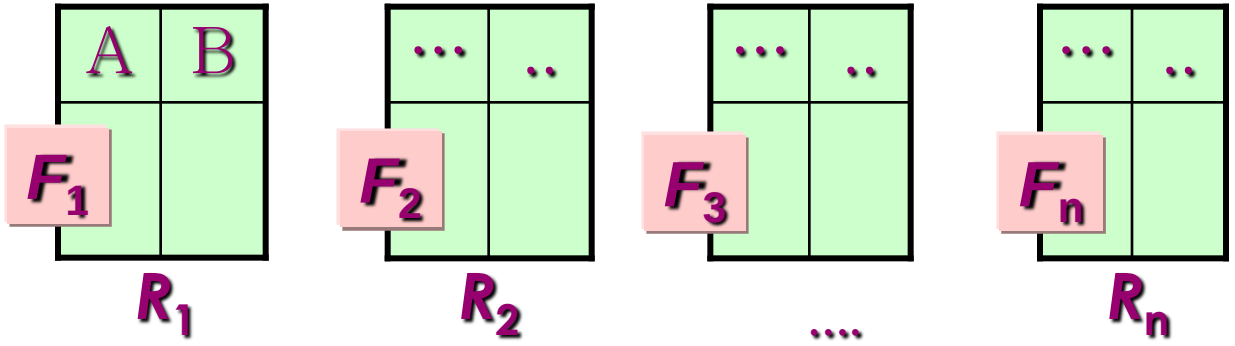
- When we decompose a relation schema R with a set of functional dependencies F into R_1, R_2,.., R_n we want
  - Lossless-join decomposition: Otherwise information loss.
  - No redundancy: R_i should be in either Boyce-Codd Normal Form or Third Normal Form.
  - Dependency preservation: the decomposition should be dependency preserving

Dependency preservation

- *Before*



- *After decompose R into* $R_1, R_2, .., R_n$



- **dependency preserving**

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

Example

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

R

A	B	C

$F = \{A \rightarrow B, B \rightarrow C\}$

- *First Method*

A	B

R_1

B	C

R_2

$F_2 = \{B \rightarrow C\}$

$F_1 = \{A \rightarrow B\}$

- Lossless-join decomposition:

$R_1 \cap R_2 = \{B\}$ and $B \rightarrow BC$

- **Dependency preserving**

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

Example

- $R = (A, B, C)$

R

A	B	C

$F = \{A \rightarrow B,\ B \rightarrow C\}$

(cannot check $B \rightarrow C$ without computing $R_1 \bowtie R_2$)

in t… …ferent ways

- *Second Method*

A	B

R_1

A	C

R_2

$F_2 = \{A \rightarrow C\}$

$F_1 = \{A \rightarrow B\}$

- Lossless-join decomposition:

$R_1 \cap R_2 = \{A\}$ and $A \rightarrow B$

- No dependency preserving

$(F_1 \cup F_2)^+ \neq F^+$

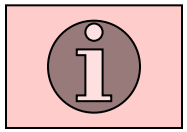
Boyce-Codd Normal Form

A relation schema *R* is in BCNF with respect to a set *F* of FD if for all FDs in F^+ of the form $Y \rightarrow X$,
where $Y \subseteq R$ and $X \subseteq R$,

at least one of the following holds:

- $Y \rightarrow X$ is trivial (i.e., $X \subseteq Y$) หรือ
- Y is a superkey for *R*

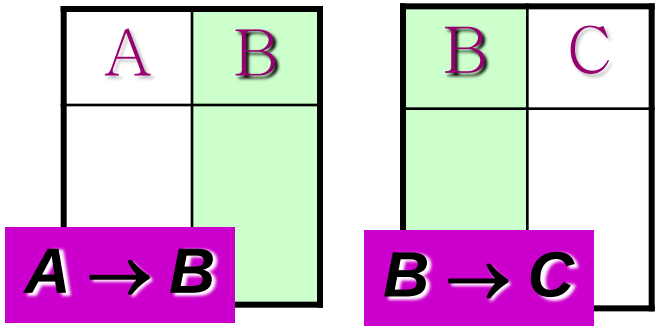
all determinants are keys.



BCNF

Example

- $R = (A, B, C)$
  $F = \{A \rightarrow B, B \rightarrow C\}$
  Key = $\{A\}$
- R is not in *BCNF* (*B is not a key but* $B \rightarrow C$)
- **Decomposition $R_1 = (A, B)$, $R_2 = (B, C)$**
  - **R_1 and R_2 in BCNF**
  - **Lossless-join decomposition**
  - **Dependency preserving**



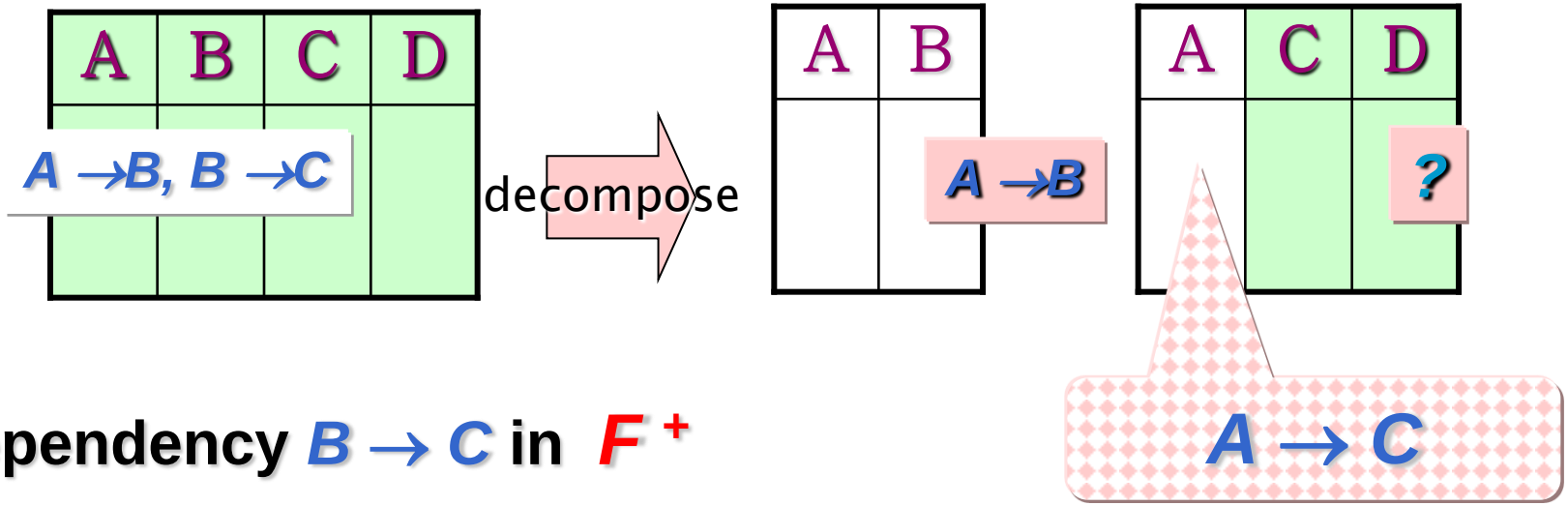
Testing for BCNF

- To check if a non-trivial dependency $X \rightarrow Y$ causes a violation of BCNF
  1. **compute X^+ and**
  2. **verify that X^+ includes all attributes of R, that is, X is a superkey of R.**

- check **only** the dependencies in the given set F for violation of BCNF, rather than checking all dependencies in F^+.

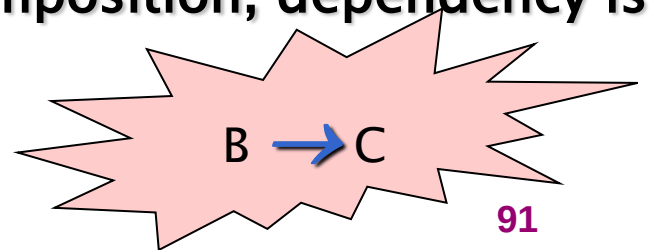
Testing for BCNF

- However, using only F is **incorrect** when testing a relation in a **decomposition** of R
  - E.g. Consider $R(A, B, C, D)$, with $F = \{ A \rightarrow B, B \rightarrow C \}$
    - Decompose R into $R_1(A,B)$ and $R_2(A,C,D)$



- **dependency $B \rightarrow C$ in F^+**

After decomposition, dependency is not preserved



BCNF Decomposition Algorithm

result := {*R*};

done := false;

compute F^+;

Start with R not in BCNF

F^+ on R

Check for FD that X is not a key

```
while (not done) do
  if (there is a schema Ri in result that is not in BCNF)
  then
    begin
        let X → Y be a nontrivial FD on Ri
        result := (result – Ri) ∪ {X, Y} ∪ (Ri – Y);
    end
  else done := true;
```

Remove R_i with problem

Add a schema in BCNF

Add a schema: get rid of Y but keep X

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

Example of BCNF Decomposition

- $R = (A, B, C, D, E, F)$

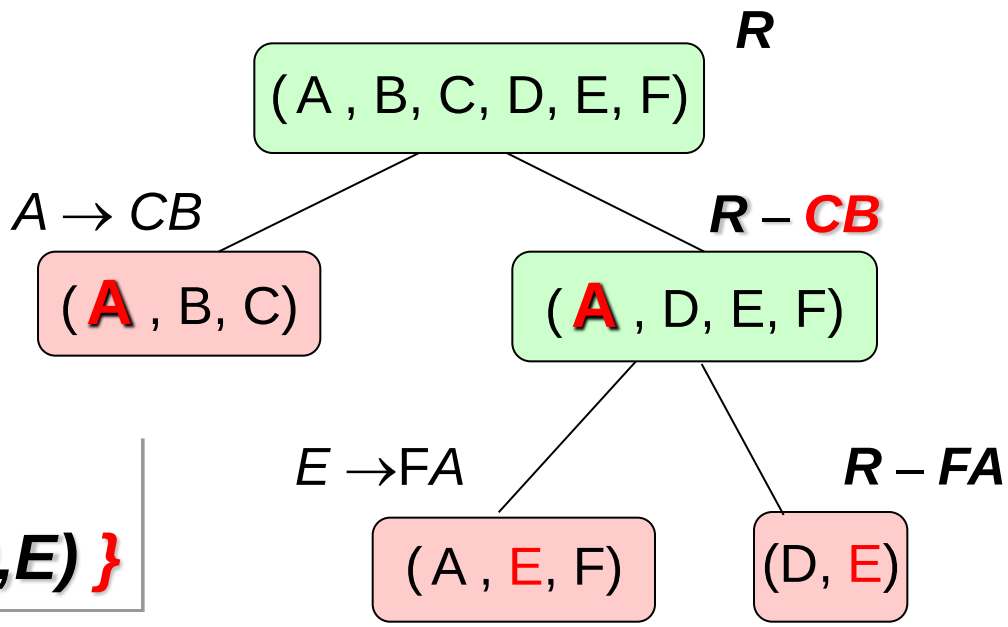
$F = \{A \rightarrow CB,\ E \rightarrow FA\}$

compute F+ by
$A^+ \rightarrow ACB$
$E^+ \rightarrow FACB$
$(ED)^+ \rightarrow FACBD$

Key = $\{E, D\}$

Not in BCNF because
A nor E is not a key

- Decomposition



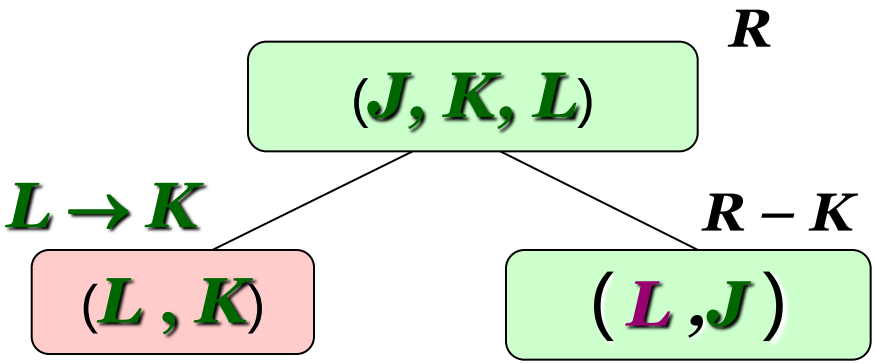
Final decomposition
{ (A,B,C), (A,E,F), (D,E) }

Put the leave nodes in the result

BCNF and Dependency Preservation

It is not always possible to get a BCNF decomposition that is dependency preserving

- $R = (J, K, L)$
  $F = \{JK \rightarrow L, L \rightarrow K\}$
  Two candidate keys = JK and JL
- R is not in BCNF (∈ L-->K but L is not a key)
- Any decomposition of R will fail to preserve $JK \rightarrow L$



Question ?