

COMP9120

Week 8: Schema Refinement and Normalisation

Semester 1, 2025

Professor Athman Bouguettaya
School of Computer Science





Acknowledgement of Country

I would like to acknowledge the Traditional Owners of Australia and recognise their continuing connection to land, water and culture. I am currently on the land of the Gadigal people of the Eora nation and pay my respects to their Elders, past, present and emerging.

Warming up



THE UNIVERSITY OF
SYDNEY

COMMONWEALTH OF AUSTRALIA

Copyright Regulations 1969

WARNING

This material has been reproduced and communicated to you by or on behalf of the University of Sydney pursuant to Part VB of the Copyright Act 1968 (the Act).

The material in this communication may be subject to copyright under the Act. Any further copying or communication of this material by you may be the subject of copyright protection under the Act.

Do not remove this notice.

- › Released **tonight**
- › **Focus:** Database Application Development
 - Gain practical experience in interacting with a relational database management system using an Application Programming Interface (API) (Python DB-API).
 - This assignment additionally provides an opportunity to use more advanced features of a database such as functions.
 - **Choice:** Java or Python
- › **Deadline:** Sunday 18 May 2025 23:59 (1 month)

› Redundancy

- Update/Insertion/Deletion anomalies

› Functional Dependencies and Normal Forms

- Functional dependencies
- Attribute closure, candidate keys
- 1NF, 2NF, 3NF, BCNF
- Multivalued dependencies and 4NF

› Schema Decomposition

- How to decompose a relation into BCNF relations
 - Lossless decomposition
 - Dependency-preservation

› Redundancy

- Update/Insertion/Deletion anomalies

› Functional Dependencies and Normal Forms

- Functional dependencies
- Attribute closure, candidate keys
- 1NF, 2NF, 3NF, BCNF
- Multivalued dependencies and 4NF

› Schema Decomposition

- Dependency-preservation and Lossless decomposition into BCNF

Redundant Data Can Cause Anomalies

Student	UnitOfStudy	Room
Mary	COMP9120	
Joe	COMP9120	
Sam	COMP9120	
..

For example

If a unit of study is in only one single room, this table contains **redundant** information!

COMP9120 only in Room 1

Student	UnitOfStudy	Room
Mary	COMP9120	R203
Joe	COMP9120	R101
Sam	COMP9120	R203
..

Forgot update this one

If we update the room number for all but one tuple, we get inconsistent data => an **update anomaly**

If everyone drops the unit of study, we lose what room this unit is in => a **delete anomaly**

Similarly, we cannot reserve a room for a unit of study without students => an **insert anomaly**

Redundant Data Can Cause Anomalies

Alternatively:

solution



Student	UnitOfStudy
Mary	COMP9120
Joe	COMP9120
Sam	COMP9120
..	..

UnitOfStudy	Room
COMP9120	R101
..	..

Is this better in terms of

- Redundancy?
- Update anomaly?
- Delete anomaly?
- Insert anomaly?

YES!

› **Bad design** usually means

Redundancy (waste of disk space)

- *Update/insertion/deletion anomalies*

› **Good design** usually means

Minimal redundancy

- *No update/insertion/deletion anomalies*

What is the **Aim** of **Good Design**?

1. Develop a **theory** to understand whether a relation must be **decomposed** to address the redundancy and update/insertion/deletion anomalies.



2. Provide a **foundational framework** on how to **decompose** the relation.

› Redundancy

- Update/Insertion/deletion anomalies

› Functional Dependencies and Normal Forms

- Functional dependencies

- Attribute closure, candidate keys
- 1NF, 2NF, 3NF, BCNF
- Multivalued dependencies and 4NF

› Schema Decomposition

- Dependency-preservation and Lossless decomposition into BCNF

Solution to bad design: *Normalize* the design to avoid the mentioned *anomalies*.

Normalization is the process that defines what is *acceptable as good relational design*.

- Geared towards resolving issues surrounding *changes* (*updates, insertion, deletions*) to the database

Before learning about the normalization process, we need to define a *key effective tool* we will use:

- **Functional Dependencies (FDs)** - a tool to:
 - Capture semantic relationships between attributes
 - Detect and eliminate bad design

In database design:

- { we use **FDs** to *identify redundancies*.
- we **also** use **FDs** to *decompose* relations to *eliminate* the related *update/insertion/delete* issues.

→ usage of FDs

› Informal definition of a Functional Dependency (FD):

Value of attribute X determines the value of attribute Y

- Every UoS is taught in a single room
 - E.g., COMP9120 is always held in Room R101

› Formal definition of a Functional Dependency (FD):

$$X \rightarrow Y$$

and we say “X (functionally) determines Y”

- Assuming that X and Y are two sets of attributes, the relationship between X and Y values is modelled using a *function*.
 - A *function* essentially means that *a value of X cannot be mapped to more than one value of Y*. Only (n-1) (and thus (1-1)) relationships may exist between X and Y.
 - Example: UoS → Room



› How do we determine **functional dependencies**?

› Two ways

- By considering:

- *Semantic meaning* of the attributes (most common)

or

- *Actual data* in tables

› In most cases, we use the *former* (i.e., *semantic meaning* of attributes).

› In the *latter* case (considering *data*), the process is called **knowledge mining**.

- Let us now look at how *data* can be used to derive FDs in the relation *Lending*.

branch_name	assests	city	loan_no	customer_name	amount
Mall St	9000000	Sydney	17	Jones	1000
Logan	5000000	Melbourne	23	Smith	2000
Queen	500000	Perth	15	Hayes	1500
Mall St	9000000	Sydney	14	Jackson	1500
King George	10000000	Brisbane	93	Carry	500
Queen	500000	Perth	25	Glenn	2500
Bondi	15000000	Adelaide	10	Brooks	2500
Logan	5000000	Melbourne	30	Johnson	750

- There is a functional dependency:

$\text{branch_name} \rightarrow \text{city}$

X ✓

branch_name	assets	city	loan_no	customer_name	amount
Mall St	9000000	Sydney	17	Jones	1000
Logan	5000000	Melbourne	23	Smith	2000
Queen	500000	Perth	15	Hayes	1500
Mall St	9000000	Sydney	14	Jackson	1500
King George	10000000	Brisbane	93	Carry	500
Queen	500000	Perth	25	Glenn	2500
Bondi	15000000	Adelaide	10	Brooks	2500
Logan	5000000	Melbourne	30	Johnson	750

› Do you see any other functional dependency?

- For instance, are the following FDs correct?

loan_no → customer_name, amount? YES

loan_no → branch_name? YES

city → customer_name? NO

city → assets? YES

sydney *Jackson* *Jones*

Value are unique

Consider the following table called *PhD*:

SID	first	last	dept	advisor	award	description
1234	Brian	Cox	IT	Codd	Rejected	Work not deemed sufficient
3456	Albert	Einstein	IT	Boyce	Conditional	Accepted with minor corrections
3456	Albert	Einstein	Physics	Newton	Accepted	Accepted with no corrections
7546	Alan	Turing	IT	Codd	Accepted	Accepted with no corrections
4879	Brian	Cox	Physics	Newton	Conditional	Accepted with minor corrections
4879	Brian	Cox	Media	Attenborough	Accepted	Accepted with no corrections

What functional dependencies might exist in the **PhD relation** above?

$Sid \rightarrow name$

$\star Brian\ Cox \rightarrow Sid$ is not

$dept \rightarrow advisor$

$award \rightarrow description$ ✓

$description \rightarrow award$ ✓

› Note that

- We might be able to tell that a certain FD does **not** hold by just looking at an instance of a relation
- However, we can **never** deduce that an FD *does* hold by looking at any number of instances of a relation.
- Why?
 - Because an FD is a statement about *all possible legal instances* of the relation.

↓ because

Armstrong's Axioms (A, B, C are sets of attributes):

Axiom definition: *An axiom is a statement that is taken to be true and serves as a premise or starting point for further reasoning and arguments¹.*

1. Reflexivity:

If $B \subseteq A$, then $A \rightarrow B$

- Example: $cpoints, uos_name \rightarrow uos_name$

These are called **Trivial FDs**: When all RHS attributes appear on LHS. All reflexive FDs are trivial.

2. Augmentation:

If $A \rightarrow B$, then $AC \rightarrow BC$ for any set of attributes C

- Example: $cpoints \rightarrow wload$ implies $cpoints, uos_name \rightarrow wload, uos_name$

3. Transitivity:

If $A \rightarrow B$ and $B \rightarrow C$, then $A \rightarrow C$

- Example: $uos_code \rightarrow cpoints, cpoints \rightarrow wload$ implies $uos_code \rightarrow wload$

¹Wikipedia

› Example

Products

Name	Color	Category	Dept	Price
Gizmo	Green	Gadget	Toys	49
Widget	Black	Gadget	Toys	59
Gizmo	Green	Whatsit	Garden	99

Functional Dependencies:

$\text{Name} \rightarrow \text{Color}$ ①
 $\text{Category} \rightarrow \text{Dept}$ ②
 $\text{Color}, \text{Category} \rightarrow \text{Price}$ ③

Given the above FDs, we *deduce* that $\text{Name}, \text{Category} \rightarrow \text{Price}$ must also hold on **any instance**: Let us use *augmentation* and *transitivity* as a proof

$\text{Name}, \text{Category} \rightarrow \text{Color}$, $\text{Category} \rightarrow \text{Dept}$ ① + ② = ④
 $\text{Color}, \text{Category} \rightarrow \text{Price}$ ④ + ③ = ⑤
Augmentation

› Example

Products

Name	Color	Category	Dept	Price
Gizmo	Green	Gadget	Toys	49
Widget	Black	Gadget	Toys	59
Gizmo	Green	Whatsit	Garden	99

Functional Dependencies:

$\text{Name} \rightarrow \text{Color}$

$\text{Category} \rightarrow \text{Dept}$

$\text{Color}, \text{Category} \rightarrow \text{Price}$

Given the above FDs, we *deduce* that $\text{Name}, \text{Category} \rightarrow \text{Price}$ must also hold on **any instance**: Let us use *augmentation* and *transitivity* as a proof

Name, Category \rightarrow Price

(4+5)

Transitivity

- › **Closure** of a set of attributes ~~FDs~~ F is the set F^+ :
 - the set of all FDs that can be deduced from the set F using the inference rules (Armstrong's axioms): reflexivity, augmentation, and transitivity inference rules.
- › More formally, the closure of a **set F** is the **set F^+** defined by:
$$F^+ = \{X \rightarrow Y \mid F \models X \rightarrow Y\}$$
- › In general, the calculation of the closure of F is, in the worst case, *exponential* in the number of attributes in F .



Algorithm for finding the **closure** of a set of FDs:

Initialize $F^+ = F$

Repeat

For each functional dependency FD in F^+

*Apply **reflexivity** and **augmentation** rules on F^+*

Add the result to F^+

For each pair of functional dependencies F_1 and F_2 in F^+

*If F_1 and F_2 can be combined using **Transitivity***

Add the result to F^+

Until F^+ does not change.

Assume that we have three attributes A, B, C in a relation R.

- With the following FDs, $F = \{A \rightarrow B, B \rightarrow C\}$
- Using the previous algorithm, F^+ includes the following FDs:

include

$A \rightarrow A,$	$AB \rightarrow A,$	$ABC \rightarrow A,$
$A \rightarrow B,$	$AB \rightarrow B,$	$ABC \rightarrow B,$
$A \rightarrow C,$	$AB \rightarrow C,$	$ABC \rightarrow C,$
$B \rightarrow B,$	$AC \rightarrow A,$ etc
$B \rightarrow C,$	$AC \rightarrow B,$	
$C \rightarrow C$	$AC \rightarrow C,$	
	$BC \rightarrow B,$	
	$BC \rightarrow C$	
.....		

- › A few additional rules (that follow from Armstrong's Axioms):

- **Decomposition**

If $A \rightarrow BC$, then $A \rightarrow B$ and $A \rightarrow C$

- **Union**

If $A \rightarrow B$ and $A \rightarrow C$, then $A \rightarrow BC$

Proof:

$A \rightarrow BC$

$BC \rightarrow B$ [Reflexivity]

$A \rightarrow B$ [Transitivity]

$A \rightarrow BC$

$BC \rightarrow C$ [Reflexivity]

$A \rightarrow C$ [Transitivity]

- › Note that 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^+

Proof:

$A \rightarrow B$

$AA \rightarrow AB$ [Augmentation]

$A \rightarrow AB$ [definition of a set, i.e., $A=AA$]

and

$A \rightarrow BC$ [Transitivity]

$A \rightarrow C$

$AB \rightarrow BC$ [Augmentation]

› Redundancy

- Update/Insertion/deletion anomalies

› Functional Dependencies and Normal Forms

- Functional dependencies
- **Attribute closure, candidate keys**
- 1NF, 2NF, 3NF, BCNF
- Multivalued dependencies and 4NF

› Schema Decomposition

- Dependency-preservation and Lossless decomposition into BCNF

- › A **superkey** is a set of attributes that *uniquely* identify each tuple in a relation
 - If K is a **superkey** for a relation R , then $K \rightarrow R$ i.e., K **determines all** the attributes of R
- › A **candidate key** (or just **key**) is a minimal superkey, i.e.,
 - **No subset** of a candidate key is a candidate key itself
 - There may be **many candidate** keys for a relation, but **only 1 primary key**.
- › Example: Given a relation R , with attributes **ABCDE** (each letter denotes an attribute) where:
 - **A** *uniquely identifies* each row in R *A is super key*
 - **BC** also *uniquely identifies* each row in R (but *not B or C alone*) *B ~~and~~ C is Candidate key*
 - **A** is a superkey (*and candidate key*) for R
 - **BC** is a superkey (*and candidate key*) for R
 - **BCE** is a superkey (*but not a candidate key*)
 - because it is **not** minimal!

› **Attribute Closure** of a set of attributes X denoted as X^+ :

- › Given a set F of FDs and a set of attributes X , X^+ (*called closure of X*) is the *set of all attributes that are determined by X under F* . It is denoted

$$X \rightarrow X^+$$

- › Attribute closure helps to *check* whether an attribute (or a set of attributes) is a **key** for the relation.

› How?

1. Any set of attributes, whose **attribute closure** is the whole relation, is a **superkey**
2. A **superkey** is a *candidate key* if it is **minimal** (i.e., **none** of its subset is a superkey)

Compute Attribute Closure: Algorithm

Starting with a given set of attributes X , we repeatedly expand the set by adding the *right side* of an FD as soon as the *left side* is present:

Algorithm to compute the **closure X^+ of a set of attribute X** :

1. Initialise *result* with the *given set* of attributes, i.e., $X^+ = \{A_1, \dots, A_n\}$ (*reflexivity rule*)
2. Repeatedly search for some *FD* $A_1 A_2 \dots A_m \rightarrow C$ such that all A_1, \dots, A_m are already in the set of attributes *result*, but C is *not*.
3. Add C to the set *result*. (*transitivity and decomposition rules*)
4. Repeat steps 2-3 until *no more attributes* can be added to *result*
5. The set *result* is then the correct value of X^+

Exercise: Candidate Keys*

What are the **keys** of the following relation:

$\text{PhD}(\text{sid}, \text{first}, \text{last}, \text{dept}, \text{advisor}, \text{award}, \text{description})$

Given the FDs

- a) $\text{sid} \rightarrow \text{first}, \text{last}$
- b) $\text{advisor} \rightarrow \text{dept}$
- c) $\text{description} \rightarrow \text{award}$
- d) $\text{sid}, \text{dept} \rightarrow \text{advisor}, \text{description}$

Let's try $(\text{sid}, \text{dept})$?

What about $(\text{sid}, \text{description})$? Is it a candidate key?

$(\text{sid}, \text{dept})^+ \rightarrow \text{sid, dept}$

$\rightarrow \text{sid, dept, first, last}$ (a)

$\rightarrow \text{sid, dept, first, last, advisor, description}$ (d)

$\rightarrow \text{sid, dept, first, last, advisor, description, award}$ (C)

$(\text{sid}, \text{dept})^+ = R$, therefore, it is a superkey.

Is it a *candidate key*? Let's check its subsets.

$\{ \text{sid} \}^+ = \{ \text{sid, first, last} \}$ **not key**

$\{ \text{dept} \}^+ = \{ \text{dept} \}$ **not key**

YES, $(\text{sid}, \text{dept})$ is a candidate key.

Short break

please stand up and stretch

Let us also menti....



THE UNIVERSITY OF
SYDNEY

› Redundancy

- Update/Insertion/deletion anomalies

› Functional Dependencies and Normal Forms

- Functional dependencies
- Attribute closure, candidate keys
- **1NF, 2NF, 3NF, BCNF**
- Multivalued dependencies and 4NF

› Schema Decomposition

- Dependency-preservation and Lossless decomposition into BCNF

Normal Forms

- The goal of normal forms is to reduce different types of redundancies
- Note: In what follows, a **key** refers to a **candidate key**, unless otherwise stated. Without loss of generality, we assume that we have **one single candidate key** in all examples, unless specified otherwise.
- Each *normal form* is characterized by a specific set of conditions.
 - For a relation to be in *a normal form*, it must satisfy the conditions associated with that form.

› We focus on **1NF, 2NF, 3NF, BCNF, and 4NF**

- Based on **Functional Dependencies (FDs)** and **MultiValued Dependencies (MVDs)**.

› **Normalisation** is the process of *meeting a design goal* which may require *breaking down* relations into smaller relations to *reduce specific types of redundancies*

- › A relation R is in **First Normal Form (1NF)** if the *domains* of *all attributes* of R are atomic.
 - *multivalued attributes* are an example of *non-atomic* domains

Important note: Objective of 1NF is ***not the same*** as the other normal forms:

According to its creator, Ted Codd, the *goal is to allow data to be queried and manipulated using a "universal data sub-language" that is grounded in first-order logic.*

Student	UnitOfStudy
Mary	{COMP9120,COMP5318}
Joe	{COMP9120,COMP5313}

Student	UnitOfStudy
Mary	COMP9120
Mary	COMP5318
Joe	COMP9120
Joe	COMP5313

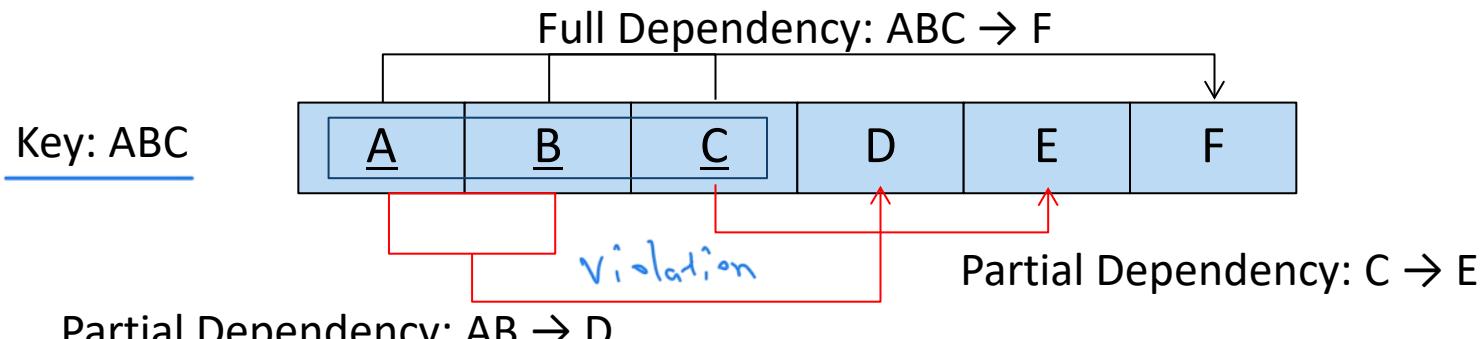
Violates 1NF

In 1NF

Second Normal Form (2NF)

› 1NF + *No partial dependencies*

- A **partial dependency** is a *non-trivial FD $X \rightarrow Y$ in R where X is a *strict (proper) subset* of some key for R and Y is not part of a key:*
- *There cannot be a functional dependency between a subset of a key to non-key attributes.* This is applicable only when the key consists of *more than one attribute*.



Teacher_name	UnitOfStudy	Teacher_position
Mary	COMP9120	Lecturer
Mary	COMP5313	Lecturer

Violates 2NF

Second Normal Form (2NF)

- › Problem: redundancy.

wed for

In the example above, the *teacher position* would have to be **repeated for all units of study that the teacher teaches!**

- › More formally,

A relation is in the 2NF if the closure F^+ contains **no** functional dependency of the form: $X \rightarrow Y$ where Y is *nonprime* (i.e., not part of a candidate key) and X is a *proper subset* of a candidate key. In this case, Y is said to be *fully functionally dependent* on the key.

- › What is the possible solution to the above *violation*?

Decompose the relation into *two relations* such that X and Y are in one relation and X is in the remaining relation. The next step is to check that the resulting relations are in 2NF.

Example above: Decompose $R(\text{Teach_name}, \text{UnitOfStudy}, \text{Teacher_position})$ into two relations: $R1(\text{Teacher_name}, \text{Teacher_position})$ and

$R2(\text{Teacher_name}, \text{UnitOfStudy})$

$\text{Teacher_name} \rightarrow \text{Teacher_position}$

split

› **Third Normal Form (3NF)**

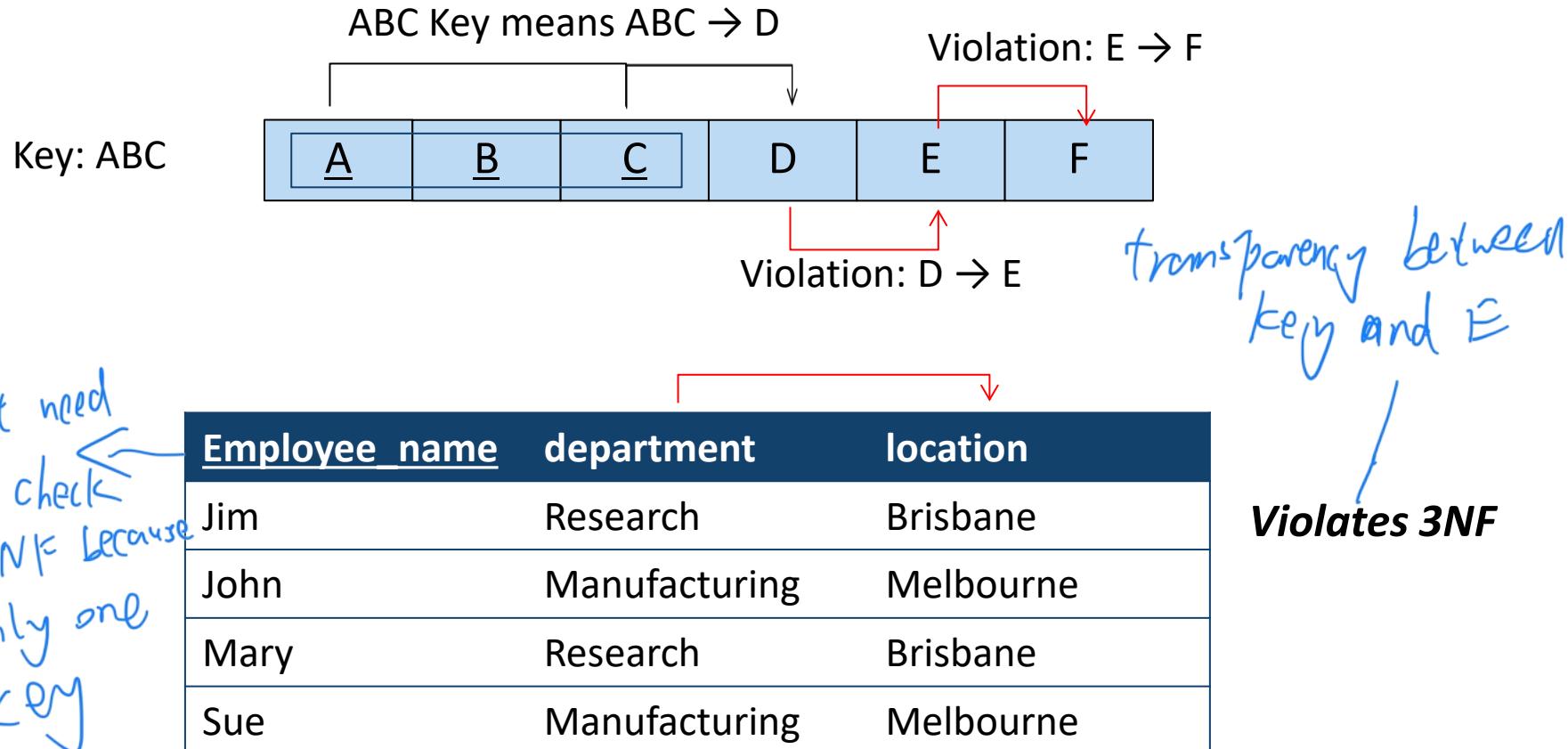
› Formal definition: a relation R is in 3NF if for each dependency $X \rightarrow Y$ in F^+ , at least one of the following conditions holds:

- $X \rightarrow Y$ is a trivial FD ($Y \subseteq X$)
- X is a superkey for R
- $Y \subset$ (is a proper subset of) a candidate key for R

› A plain English definition of 3NF is: A relation is in 3NF if and only if (denoted iff) it is in 2NF and every non-key attribute is not transitively functionally dependent on a key.



new (compare with 2NF)



- There is a *functional dependency* between *Department* and *Location* (thus *transitive dependency*).



- › Problem: redundancy

The department's location is *repeated* with *every employee* working in that department.

- › Solution: split up the relation into two relations:

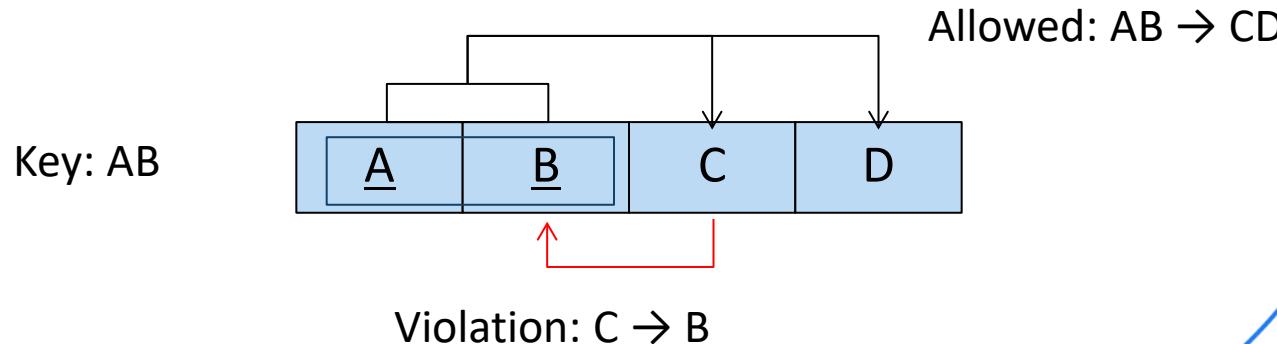
- R1(Employee, Department) and R2(Department, Location)

<u>Employee</u>	<u>Department</u>
<u>Department</u>	<u>Location</u>

- › Boyce-Codd Normal Form (BCNF) is a **stronger form** of 3NF

Problem: 3NF *allows functional dependencies between non-prime attributes to prime attributes.*

- › A relation R is in **BCNF** if **all non-trivial FDs over R , have** a superkey of R on the **Left Hand Side (LHS)**. In other words, a relation is in BCNF *iff every attribute is dependent on the key, the whole key and nothing but the key.*
- Formally: For all *non-trivial* $X \rightarrow Y$ for R : X is a superkey for R
- Informally: All dependency arrows are from superkeys



- › Boyce-Codd Normal Form (BCNF) is a **stronger form** of 3NF

Problem: 3NF *allows functional dependencies between non-prime attributes to prime attributes.*

- › A relation R is in **BCNF** if **all non-trivial FDs over R , have** a superkey of R on the Left Hand Side (LHS). In other words, a relation is in BCNF *iff* every attribute *is dependent on the key, the whole key and nothing but the key.*
 - Formally: For all *non-trivial* $X \rightarrow Y$ for R : X is a superkey for R
 - Informally: *All dependency arrows come out of superkeys*



A red bracket with a downward-pointing arrow spans the first two columns ('Teacher_name' and 'UnitOfStudy') of a table. To the right of the bracket, the text '走向不键' (Non-key dependency) is written in blue.

<u>Teacher_name</u>	<u>UnitOfStudy</u>	Address
Mary	COMP9120	One Street
Mary	COMP5313	One Street

Violation: Address \rightarrow Teacher_name

› Problem: redundancy

The teacher name is repeated for every address of a teacher that teaches a unit of study.

› Solution: split up the relation into two relations:

- R1(Address, Teacher_name) and R2(Address, UnitofStudy)

<u>Address</u>	Teacher_name
----------------	--------------

<u>UnitOfStudy</u>	Address
--------------------	---------



Enrol(sid, uosCode, title, credits, hours, lecturer, grade)

Key = (sid, uosCode)

1. $\text{uosCode} \rightarrow \text{title, credits, lecturer}$
2. $\text{credits} \rightarrow \text{hours}$

NO:

1. violates 2NF, therefore it violates BCNF
2. violates 3NF, therefore it violates BCNF

› Redundancy

- Update/Insertion/deletion anomalies

› Functional Dependencies and Normal Forms

- Functional dependencies
- Attribute closure, candidate keys
- 1NF, 2NF, 3NF, BCNF
- **Multivalued dependencies and 4NF**

› Schema Decomposition

- Dependency-preservation and Lossless decomposition into BCNF

Example of mvd

Issues with 1NF

- Because the 1NF *does not allow more than one value* for each attribute, here is an example of how we can deal with *multiple* values for an attribute.

name	profession	Language
John	{Electrician, Plumber}	French, Korean
Mary	{Doctor, Author}	Spanish, Chinese

name	profession	Language
John	Electrician	French
Mary	Doctor	Spanish
John	Plumber	Korean
Mary	Author	Chinese

Still have problem

- Because the 1NF *does not allow more than one value* for each attribute, here is an example of how we can deal with *multiple* values for an attribute.

However, these values suggest that whenever

means

- John is electrician, he speaks French.
- John is plumber, he speaks Korean.

Same for Mary. The values suggest that whenever

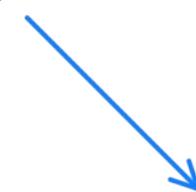
- Mary is an author, she speaks Chinese
- Mary is a doctor, she speaks Spanish

This table data is implying that there is a relationship between the *Profession* John is engaged in **and** the *Language* that he speaks!

This is of course semantically incorrect. John and Mary speak these languages irrespective of the professions they are engaged in!

name	profession	Language
John	Electrician	French
Mary	Doctor	Spanish
John	Plumber	Korean
Mary	Author	Chinese

*So **not to infer** such an incorrect semantic relationship, there is a need to repeat all combinations of these attributes.*



This means that

John will have both languages occurring with both of his professions!

Mary will have both languages occurring with both of her professions!

name	profession	Language
John	Electrician	French
Mary	Doctor	Spanish
John	Plumber	Korean
Mary	Author	Chinese

name	profession	Language
John	Electrician	French
Mary	Doctor	Spanish
John	Plumber	Korean
Mary	Author	Chinese
John	Plumber	French
Mary	Author	Spanish
John	Electrician	Korean
Mary	Doctor	Chinese

- Informal definition: A **multivalued dependency** between X and Y exists if no relationship can be inferred between X and Z, i.e., they are *independent* of each other.
- More formally, let R (X, Y, Z) be a relation and X, Y and Z are sets of attributes of R, there is a *multivalued dependency*, noted:

$$X \twoheadrightarrow Y \text{ (X multidermines Y)}$$

if for all tuples t_1 and t_2 that agree in X, i.e., $t_1[X] = t_2[X]$, there exist tuples t_3 and t_4 such that:

1. $t_1[X] = t_2[X] = t_3[X] = t_4[X]$ and
2. $t_3[Y] = t_1[Y]$ and $t_4[Y] = t_2[Y]$ and
3. $t_3[Z] = t_2[Z]$ and $t_4[Z] = t_1[Z]$

tuples	X	Y	Z
t_1	$a_1 \dots a_i$	$a_{i+1} \dots a_j$	$a_{j+1} \dots a_n$
t_2	$a_1 \dots a_i$	$b_{i+1} \dots b_j$	$b_{j+1} \dots b_n$
t_3	$a_1 \dots a_i$	$a_{i+1} \dots a_j$	$b_{j+1} \dots b_n$
t_4	$a_1 \dots a_i$	$b_{i+1} \dots b_j$	$a_{j+1} \dots a_n$

MultiValued Dependency (MVD): Example

Example:

$UoS \Rightarrow Textbook$?

Note that according to the values in the relation, the relationship between the UoS and Textbook is **independent** from the relationship between UoS and Tutor. This means that there is an MVD between *UoS* and *Textbook*. This also implies that the textbook of a *UoS* is set *independently*.

Assume a new textbook by Ramakrishnan is added to the UoS COMP9120. What should happen to maintain this independence (i.e., MVD)?

- Add one row for each Tutor of that UoS.

<u>UoS</u>	<u>Textbook</u>	<u>Tutor</u>
COMP9120	Silberschatz	Ying Z
COMP9120	Widom	Ying Z
COMP9120	Silberschatz	Mohammad P
COMP9120	Widom	Mohammad P
COMP9120	Silberschatz	Alan F
COMP9120	Widom	Alan F
COMP5110	Silberschatz	Ying Z
COMP5110	Silberschatz	Mohammad P
COMP9120	Ramakrishnan	Alan F
COMP9120	Ramakrishnan	Ying Z
COMP9120	Ramakrishnan	Mohammad P

still have problem

Fourth Normal Form (4NF)

Redundancy problem in MVDs:

- For the first example: should list *all professions* for *every language* a person speaks.
- For the second example: should *list all tutors* for *each textbook* that a UoS has listed.

4NF deals with *redundancies* created by ***multivalued dependencies***.

Formally:

R is in **4NF** if for all **MVDs** of the form $X \twoheadrightarrow Y$ in F^+ , at least one of the following conditions holds:

- $X \twoheadrightarrow Y$ is a *trivial MVD* (i.e., either $Y \subseteq X$ or $X \cup Y = R$)
- X is a *superkey* for R

? X is a set
 Y is a set
R is a set

Assuming the *only key* to the following relation is the set : (Project-id, Personal-phone#):

<u>employee_name</u>	<u>project_id</u>	<u>personal_phone_number</u>
Bob	P1	047012345
Bob	P3	046098765
Bob	P1	046098765
Bob	P3	047012345
Lily	P1	045067543
Fiona	P7	043085432

Is this relation in 4NF?



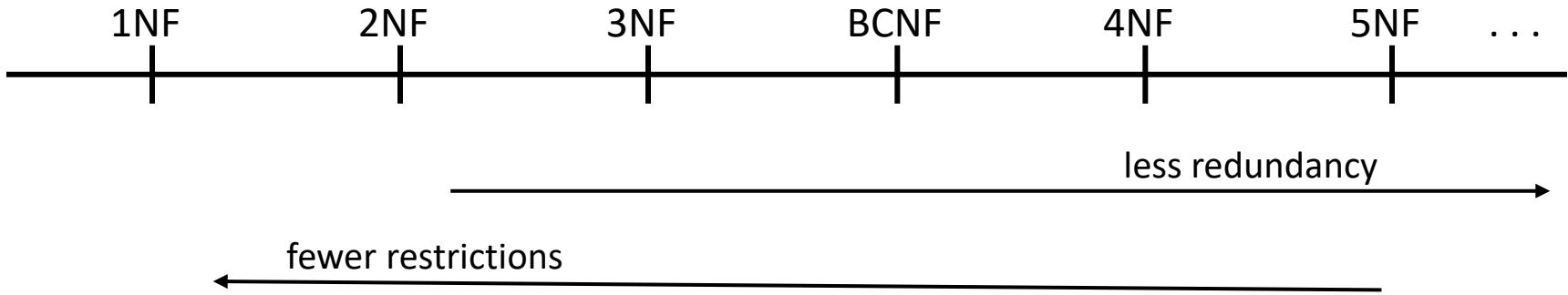
No: There is at least *one non-trivial multivalued dependency*

$\text{employee_name} \twoheadrightarrow \text{Project_id}$ (Note that employee_name is *not* a superkey).

Solution: split the above relation into two relations:

Now the two relations are in 4NF!





› Higher normal forms

- 5NF, 6NF/DKNF
- They also exploit other types of dependencies
 - Join dependencies
 - Inclusion dependencies

› Redundancy

- Update/Insertion/deletion anomalies

› Functional Dependencies and Normal Forms

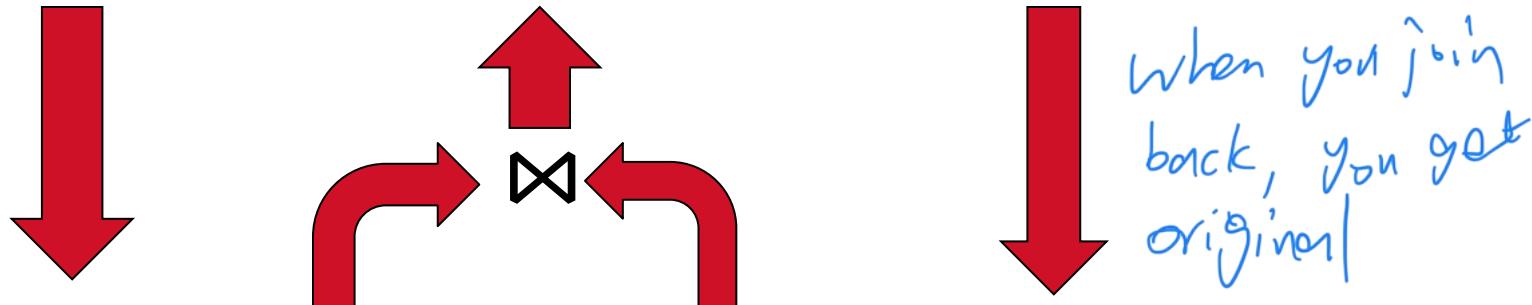
- Functional dependencies
- Attribute closure, candidate keys
- 1NF, 2NF, 3NF, BCNF
- Multivalued dependencies and 4NF

› Schema Decomposition

- ***Dependency-preservation*** and ***lossless decomposition*** into BCNF

Decomposition: Example 1

Student	UoSCode	UoSName	Grade
Alice	COMP5138	Database Management Systems	CR
Alice	COMP5338	Advanced Data Models	D
Bob	COMP5138	Database Management Systems	P
Clare	COMP5338	Advanced Data Models	HD
David	COMP5338	Advanced Data Models	CR

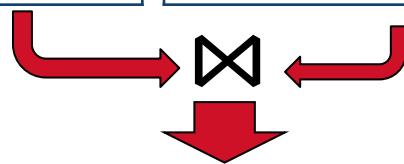


Student	UoSCode	Grade	UoSCode	UoSName
Alice	COMP5138	CR	COMP5138	Database Management Systems
Alice	COMP5338	D	COMP5338	Advanced Data Models
Bob	COMP5138	P		
Clare	COMP5338	HD		
David	COMP5338	CR		

Decomposition: Example 2

Student	UoSCode
Alice	COMP5138
Alice	COMP5338
Bob	COMP5138
Clare	COMP5338
David	COMP5338

UoSCode	UoSName	Grade
COMP5138	Database Management Systems	CR
COMP5138	Database Management Systems	P
COMP5338	Advanced Data Models	CR
COMP5338	Advanced Data Models	D
COMP5338	Advanced Data Models	HD



Student	UoSCode	UoSName	Grade
Alice	COMP5138	Database Management Systems	CR
Alice	COMP5138	Database Management Systems	P
Alice	COMP5338	Advanced Data Models	CR
Alice	COMP5338	Advanced Data Models	D
Alice	COMP5338	Advanced Data Models	HD
Bob	COMP5138	Database Management Systems	CR
Bob	COMP5138	Database Management Systems	P
Clare	COMP5338	Advanced Data Models	CR
Clare	COMP5338	Advanced Data Models	D
Clare	COMP5338	Advanced Data Models	HD
David	COMP5338	Advanced Data Models	CR
David	COMP5338	Advanced Data Models	D
David	COMP5338	Advanced Data Models	HD

lossing
 ↗ infor
 we do not
 know Alice
 pass or
 not

- › A decomposition of R **replaces** R by two or more *distinct* relations:
 - Each new relation schema contains a subset of the attributes of R, and
 - Every attribute of R appears as an attribute in *at least one* of the new relations.
 - Many possible decompositions – however, *not all equally* good/correct
- › Need strong **decomposition properties**:
 - **Dependency preservation:** *No FDs are lost* in the decomposition
 - **Lossless-join (also called non-additive) decomposition:** *Re-joining a decomposition* of R should give us back R!

› Dependency preservation

When decomposing a relation, we may require that *dependencies be preserved* in the resulting component relations because dependencies are also used to *express the constraints* on the database. If they are *not preserved*, to check whether the *dependencies still hold*, a solution would be to perform '*costly joins*'.

› Assuming a relation R is decomposed into relations $R_1 R_2 \dots R_{n-1} R_n$

First check whether a "lost" dependency *can be deduced* across relations. If it *cannot be deduced*, we have no other choice but to perform *joins* to check whether a functional dependency still holds every time we *have an update*.

› How do we check?

Let $F' = F_1 \cup F_2 \cup \dots \cup F_{n-1} \cup F_n$ be the *union* of sets of FDs of the *decomposed relations*. F_i is that subset of F' that is applicable (i.e., projection) to R_i .

If $F' \neq F$, run a simple algorithm that checks whether $F'^+ = F^+$.

decomposed set

Dependency preservation: An Example

- › Assume we have the following relation and FDs:
- › $R = (A, B, C)$, $F = \{A \rightarrow B, B \rightarrow C, A \rightarrow C\}$
- with Key = {A}

This relation is *not* in 3NF because of the transitive FD derived from $B \rightarrow C$. We therefore split the relation R into $R_1 = (B, C)$ and $R_2 = (A, B)$. Both relations *are now in 3NF* because there is no transitive FD.

However, is this decomposition dependency preserving?

Let F_1 be the projection of F' on R_1 and F_2 the projection of F' on R_2 . To be dependency preserving, the above decomposition would have to satisfy either $F' = F_1 \cup F_2$ or $F'^+ = (F_1 \cup F_2)^+$ (i.e., the *closure of F'* is equal to the *closure of $(F_1 \cup F_2)$*).

For the relation R_1 , $F_1 = \{B \rightarrow C\}$, omitting all trivial FDs. For R_2 , $F_2 = \{A \rightarrow B\}$, omitting all trivial FDs. Therefore $F \neq \{F_1 \cup F_2\}$ because we lost the FD: $A \rightarrow C$! Let us know check whether F^+ is equal $(F_1 \cup F_2)^+$. The answer is YES.

This means that this decomposition is *dependency preserving*.

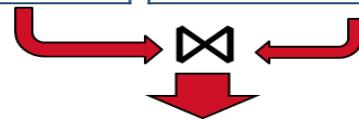
› Lossless join decomposition

The loss here refers *to the loss of information* and *not to the loss of tuples!*

When joining back the resulting relations, we *may be losing information by adding extra tuples as we saw before!* Remember the example:

Student	UoSCode
Alice	COMP5138
Alice	COMP5338
Bob	COMP5138
Clare	COMP5338
David	COMP5338

UoSCode	UoSName	Grade
COMP5138	Database Management Systems	CR
COMP5138	Database Management Systems	P
COMP5338	Advanced Data Models	CR
COMP5338	Advanced Data Models	D
COMP5338	Advanced Data Models	HD



Student	UoSCode	UoSName	Grade
Alice	COMP5138	Database Management Systems	CR
Alice	COMP5138	Database Management Systems	P
Alice	COMP5338	Advanced Data Models	CR
Alice	COMP5338	Advanced Data Models	D
Alice	COMP5338	Advanced Data Models	HD
Bob	COMP5138	Database Management Systems	CR
Bob	COMP5138	Database Management Systems	P
Clare	COMP5338	Advanced Data Models	CR
Clare	COMP5338	Advanced Data Models	D
Clare	COMP5338	Advanced Data Models	HD
David	COMP5338	Advanced Data Models	CR
David	COMP5338	Advanced Data Models	D
David	COMP5338	Advanced Data Models	HD

- › A decomposition is not always lossless. We may have *more than one choice* to *split up* a relation *in the course of normalization* as we saw in the previous example.

For example, *if the decomposition is on a non-key attribute, the decomposition may be lossy*.

To ensure lossless join decomposition use *Functional Dependencies*.

- › If R is a relation decomposed into relations $R_1, R_2, \dots, R_{n-1}, R_n$ and F is a set of functional dependencies. This *decomposition* is *lossless* (with respect to F) if for every relation R that satisfies F , the following equation is true:
 - $R = \Pi R_1(R) \bowtie \Pi R_2(R) \dots \bowtie \Pi R_{n-1}(R) \bowtie \Pi R_n(R)$ where Π and \bowtie are the symbols for *projection* and *natural join*, respectively.
- › More formally: Let R be a relation and R_1 and R_2 be a decomposition of R . Let F be the set of functional dependencies. This decomposition is a lossless-join decomposition if at least one of the following FDs are in F^+ .

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

Rule

- › In plain English, this means that if the *intersection* of the set of attributes between R_1 and R_2 functionally determines either R_1 or R_2 (i.e., the intersection is a key to at least one of the resulting relations), then the composition is *lossless*.

- › Suppose we have a schema R and a non-trivial dependency $X \rightarrow Y$ which causes a violation of BCNF. We decompose R into:

$$R_1 = X \cup Y$$

$$R_2 = R - Y$$

- › Example schema that is *not* in BCNF:

$loan_info = (\underline{customer_id}, \underline{loan_number}, amount)$ with $loan_number \rightarrow amount$

but $loan_number$ is *not a superkey*

- › Assume,

- $X = loan_number$
- $Y = amount$

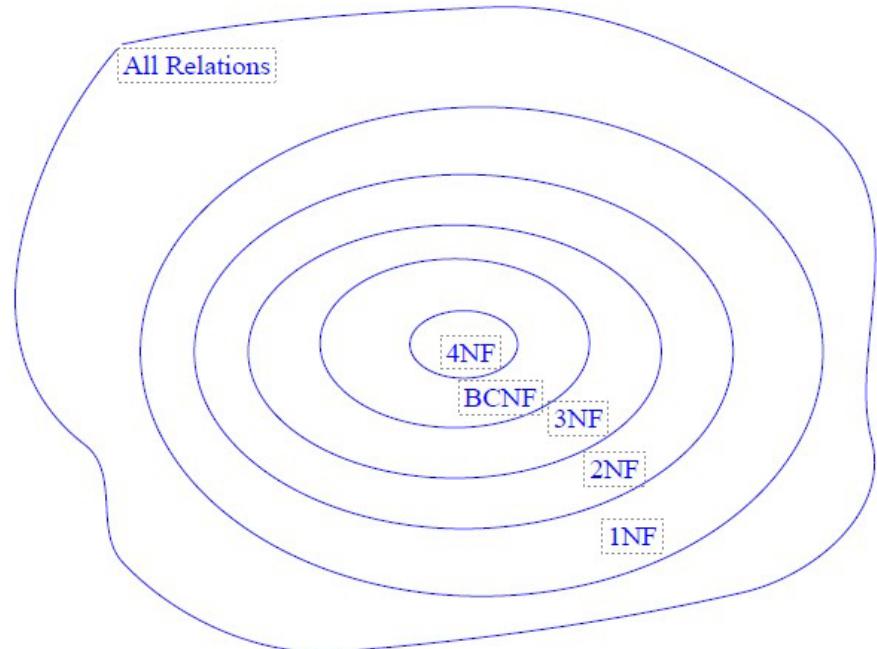
So, the relation $loan_info$ is replaced by the following relations:

$$R_1 = (X \cup Y) = (\underline{loan_number}, amount)$$

$$R_2 = (R - Y) = (\underline{customer_id}, \underline{loan_number})$$

Now both are in BCNF

Relationship between Normal Forms:



Remember that normalization is

- Geared towards changes/updates in the database
- Not necessarily good for mostly retrieval operations

This means that it is *not always good to normalize up to the 4NF normal form*: e.g., archives, historical databases, etc.

You should be able to perform the following tasks



- › Identify and interpret functional dependencies for a database schema
- › Identify the candidate keys and normal form (up to 4NF) of a relation schema using its functional and multivalued dependencies
 - Identify whether a set of attributes forms a minimal superkey
 - Determine all possible candidate keys
- › Decompose a relational instance into a set of BCNF relational instances
 - Determine a lossless join decomposition of a relational schema
 - Correctly determine the decomposed relation instances

› **Transaction Management**

- Transaction Concept
- Serializability

› **Readings:**

- **Ramakrishnan/Gehrke, Chapter 16**
- Kifer/Bernstein/Lewis book, Chapter 18
- Ullman/Widom, Chapter 6.6 onwards



Have a nice break!



THE UNIVERSITY OF
SYDNEY