COMP9120

Week 9: Schema Refinement and Normalisation

Semester 2, 2022

Professor Athman Bouguettaya
School of Computer Science

THE UNIVERSITY OF SYDNEY

Contents developed by Dr Lijun Chang

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 Darug people and pay my respects to their Elders, past, present and emerging.

I further acknowledge the Traditional Owners of the country on which you are on and pay respects to their Elders, past, present and future.

Copyright warning

Outline

- **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

Outline

- **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	R101
Joe	COMP9120	R101
Sam	COMP9120	R101
..	..	..

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

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

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 can't reserve a room for a unit of study without students = an ***insert* anomaly**

Redundant Data Can Cause Anomalies

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

UnitOfStudy	Room
COMP9120	R101
..	..

Is this better?

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

Another example

- Example: a bank database called *Lending*:

Table 1: *LENDING*

branchname	assets	city	loan	customer	amount
Mall St	9000000	LA	17	Jones	1000
Logan	5000000	SF	23	Smith	2000
Queen	500000	DC	15	Hayes	1500
Mall St	9000000	LA	14	Jackson	1500
King George	10000000	Detroit	93	Curry	500
Queen	500000	DC	25	Glenn	2500
Andrew	15000000	NY	10	Brooks	2500
Logan	5000000	SF	30	Johnson	750

- Consider the following scenario: suppose we want to insert one more entry reflecting that Harris is requesting a loan (the id is 40) of 2100 dollars from the Mall St branch:

(Mall St, 40, Harris, 2100)

- This means that we would have to insert the following tuple to *Lending*:

(Mall St, 9000000, LA, 40, Harris, 2100)

- But then the branch information is repeated *many times* in the relation *Lending*!

Redundancy issues

- What are the problems?
  - Information is *needlessly repeated*, causing:
    - Waste of space
      - *Branchname, assets,* and *city* are repeated for each new *loan* in the relation *Lending*, thus wasting disk space.
  - Potential inconsistencies due to updates, deletions and insertions
    - If *branchname* changes, there is a need to update more than one tuple. Possible inconsistencies.
    - Assume *Lending* is the *only* relation in the database so far: if a new branch is inserted and no loans have been awarded yet, how are we to record this information?
      - May be add null values but are usually hard to handle because of their ambiguous semantics.
    - If all loans have been paid out, information about the branch would also be deleted!

Bad and good database design

- **Bad design** usually means
  - Redundancy (waste of disk space)
  - Update/insertion/deletion anomalies
- **Good design** usually means
  - Minimal redundancy
  - No update/insertion/deletion anomalies

Develop a theory to understand ***whether*** a relation needs to be ***decomposed*** to address the redundancy issue and update/insertion/deletion anomalies. Provide an approach on **how** to decompose the relation.

Outline

- **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

Normalization

› Solution to bad design: *Normalize* the design to avoid these pitfalls

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

Note that *normalization* is:

- Geared towards resolving issues surrounding ***updates***

Before we talk about the normalization process, we need to define some needed tools:

- **Functional Dependencies (FDs)**: What are they?

They are a tool to:

- *capture semantic relationships* between attributes
- *detect* and *eliminate bad design*

In database design, FDs are used to *break up (i.e., decompose)* relations *and* to *detect* and *eliminate* update/insertion/deletion anomalies and redundancy.

Functional Dependencies

- **Dependency**: value of attribute X **determines** the value of attribute Y
  - Every UoS is taught in a single room
    - COMP9120 is always held in Room R101
- **Functional Dependency (FD)**:
  - Written as **X $\longrightarrow$ Y**, and we say "X (functionally) **determines** Y"
    - *Definition of a functional dependency*: Assuming that X and Y are two sets of attributes, the relationship between X and Y values is modelled using a ***function***. This 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.
    - The functional dependency is denoted, X → Y

UoS → Room

- FDs are used to identify *sources of redundancy* within schemas.

Functional Dependencies

› How do we determine functional dependencies?

› By considering either:

- *The semantic meaning* of the attributes

or

- *Data*

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

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

Another example

- Let us now look at how data can be used to derive FDs in the relation *Lending*.
- There is a functional dependency:

branchname → city

Table 1: *LENDING = BRANCH ∞ BORROW*

branchname	assets	city	loan	customer	amount
Mall St	*9000000*	*LA*	*17*	*Jones*	*1000*
Logan	*5000000*	*SF*	*23*	*Smith*	*2000*
Queen	*500000*	*DC*	*15*	*Hayes*	*1500*
Mall St	*9000000*	*LA*	*14*	*Jackson*	*1500*
King George	*10000000*	*Detroit*	*93*	*Curry*	*500*
Queen	*500000*	*DC*	*25*	*Glenn*	*2500*
Andrew	*15000000*	*NY*	*10*	*Brooks*	*2500*
Logan	*5000000*	*SF*	*30*	*Johnson*	*750*

Functional Dependencies

Table 1: *LENDING = BRANCH ∞ BORROW*

branchname	assets	city	loan	customer	amount
Mall St	*9000000*	*LA*	*17*	*Jones*	*1000*
Logan	*5000000*	*SF*	*23*	*Smith*	*2000*
Queen	*500000*	*DC*	*15*	*Hayes*	*1500*
Mall St	*9000000*	*LA*	*14*	*Jackson*	*1500*
King George	*10000000*	*Detroit*	*93*	*Curry*	*500*

Table 1: *LENDING = BRANCH ∞ BORROW*

branchname	assets	city	loan	customer	amount
Mall St	*9000000*	*LA*	*17*	*Jones*	*1000*
Logan	*5000000*	*SF*	*23*	*Smith*	*2000*
Queen	*500000*	*DC*	*15*	*Hayes*	*1500*
Mall St	*9000000*	*LA*	*14*	*Jackson*	*1500*
King George	*10000000*	*Detroit*	*93*	*Curry*	*500*
Queen	*500000*	*DC*	*25*	*Glenn*	*2500*
Andrew	*15000000*	*NY*	*10*	*Brooks*	*2500*
Logan	*5000000*	*SF*	*30*	*Johnson*	*750*

Observing FDs

- Note that
  - We might be able to tell that a certain FD does ***not*** hold by 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, because an FD **is a statement** about *all* possible *legal* instances of the relation.

FD Inference Rules

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

1. **Reflexivity:** If $B \subseteq A$, then $A \rightarrow B$

- Example: *cpoints, uos_name* $\rightarrow$ *uos_name*

2. **Augmentation:** If $A \rightarrow B$, then $AC \rightarrow BC$ for any 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*

› **Trivial FDs**: When all RHS attributes appear on LHS: *all reflexive* FDs are trivial.

Inferring Functional Dependency

› Example

Products

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

Provided FDs:

1. Name → Color
2. Category → Dept
3. Color, Category → Price

Given the provided FDs, we can see that *Name, Category → Price* must also hold on **any instance** (use augmentation and transitivity)…

Name → Color
Color, Category → Price

FD Basis and Minimality

- Let F be a set of functional dependencies:
- We say that F *logically implies* the FD X → Y, (denoted F |= X → Y) if for all functional dependencies in F that are satisfied by a relation R, this implies that the FD X → Y is also satisfied by the relation R.

Example:

- If the set F = { X → Y and Y → Z } then F |= X → Z (using the transitivity rule)

- **Basis**: A set of FDs is called a *basis* for a relation R if we can infer from this set all FDs for the relation R.
- **Minimality**: If there is **no subset** of a *basis* from which we can derive the set of all FDs for a relation R, then the *basis* is *minimal*.

Closure of a set

› *Closure* of a set F is the set F^+: It is 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.

› All FDs logically implied by an initial set of given FDs F, is known as the **closure of F, labelled F^+**

› More formally, the closure of a **set F** is the **set F^+** defined by:

$$\mathbf{F^+ = \{X \rightarrow Y \mid F \models X \rightarrow Y\}}$$

› In general, the calculation of the closure of F is, in the worst case, exponential in the number of attributes in F.

Closure of a set

- *Closure* of a set F is the set F^+: It is 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.
- Algorithm
- *$F^+ = F$*
- *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 F1 and F2 in F^+*
- *If F1 and F2 can be combined using Transitivity*
- *Add the result to F^+*
- *Until F^+ stays the same.*

FD Closure Example

- Example:
  - Assume that we have three attributes A, B, C in a relation R.
    - With the following FDs, F = {A → B, B → C}.
- Using the above algorithm, F^+ includes the following FDs: A → A, A → B, A → C , B → B, B → C , C → C , AB → A, AB → B, AB → C , AC → A, AC → B, AC → C , BC → B, BC → C , ABC → A, ABC → B, ABC → C

FD Closure

- A few additional rules (that follow from Armstrong's Axioms.):
  - **Union**

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

  - **Decomposition**

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

Proof:

$A \to B$ | $A \to C$

$AA \to \text{AB}$ [Augmentation] | $AB \to \text{BC}$

$\text{A} \to \text{AB}$

$\text{A} \to \text{BC}$ [Transitivity]

- 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^+

Outline

- **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

Functional Dependency and Keys

- A **superkey** is a set of attributes that *uniquely* identify each tuple in a relation
  - If K is a superkey for a relation R, K $\rightarrow$ R i.e., K *determines all* the attributes of R
- A **candidate key** (or just key) is a **<u>minimal</u>** superkey
  - *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.

- Given a relation R, with attributes **ABCDE** (each letter denotes an attribute) where:
  - **A** uniquely identifies each row in R
  - **BC** also uniquely identifies each row in R (but not B or C alone)
- **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)
  - …..as it is *not* minimal

THE UNIVERSITY OF SYDNEY

Attribute Closure

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

$$\boldsymbol{X \rightarrow X^+}$$

- If we know the functional dependencies, then we can check whether an attribute (or a set of attributes) is a key for the relation
  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

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

Algorithm:

1. Initialise *result* with the given set of attributes: $X=\{A_1, \ldots, A_n\}$ *(reflexivity rule)*
2. Repeatedly search for some *FD* $A_1 A_2 \ldots A_m \rightarrow C$ such that all $A_1, \ldots, A_m$ 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 step 2 until no more attributes can be added to *result*
5. The set *result* is the correct value of X^+

Exercise: Candidate Keys

What are the keys of our relation?

PhD(sid, first, last, dept, advisor, award, description)

Given FDs

a) sid → first, last

b) advisor → dept

c) description → award

d) sid, dept → advisor, description

Keys?

1. (sid, dept)? Compute (sid, dept)$^+$={sid,dep} U {advisor,description} U {first, last} U {award} = R, therefore **is a key**. Is it a candidate key? **Yes!** Why? {sid}$^+$={sid, first, last} **not key** and {dept}$^+$={dept} **not key**
2. (sid, advisor)? Left as an exercise.

Outline

- 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

- Database theory identifies several ***normal forms*** for relational schemas.
  - Normal Forms
    - Goal: reduce redundancy (hence potential anomalies)
    - 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 set of restrictions.
    - For a relation to be in a normal form, it must satisfy the restrictions associated with that form.
- We focus on **1NF, 2NF, 3NF, BCNF,** and **4NF**
  - Restrictions are based on *functional dependencies* and *multivalued dependencies*.
- Normalisation *is usually used to verify a design*
  - Start a design with a set of relations and use *normalisation* to *break them* down into smaller relations

First Normal Forms

› 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

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

Violates 1NF

Student	UnitOfStudy
Mary	COMP9120
Mary	COMP5318
Joe	COMP9120
Joe	COMP5313

In 1NF

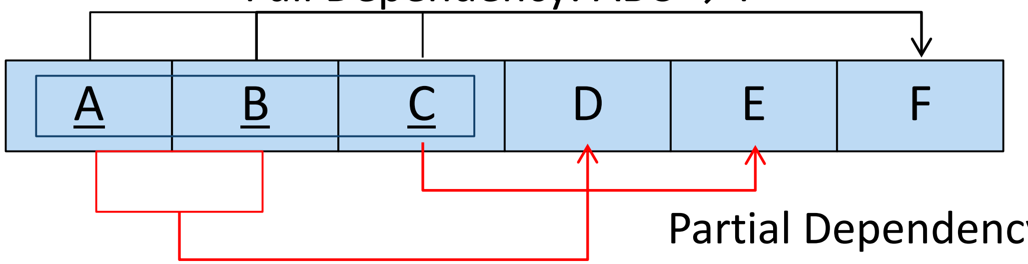
Second Normal Form (2NF)

› 1NF + **No partial dependencies**

- A *partial dependency* is a *non-trivial* FD $X \rightarrow Y$ for 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.

Full Dependency: ABC → F

Key: ABC



Partial Dependency: C → E

Partial Dependency: AB → D

<u>Teacher_id</u>	<u>UnitOfStudy</u>	Teacher_position
Mary	COMP9120	Lecturer
Mary	COMP5313	Lecturer

Violates 2NF

Second Normal Form (2NF)

- Problem: redundancy.

  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* (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.

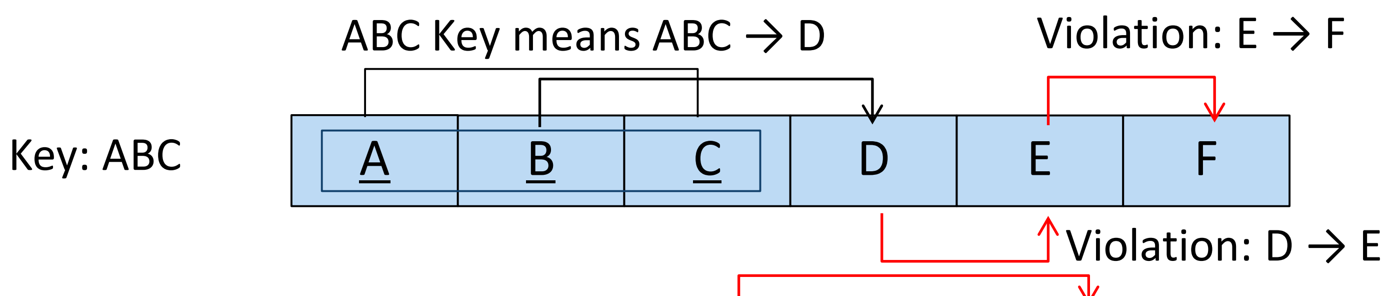
- What is the solution to the above violation?

  Decompose/split up the relation in two relations such that X and Y form one relation and X is in the remaining relation. Check that the resulting relations are in 2NF. At least, the relation (X,Y) is now in 2NF. Check the remaining relation if it is in 2NF.

- **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 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

- Another definition of 3NF is : a relation is in 3NF iff it is in 2NF and *every non-key* attribute is *not transitively* functionally dependent on a key.

Third Normal Form



<u>Employee</u>	Department	Location
Jim	Research	Brisbane
John	Manufacturing	Melbourne
Mary	Research	Brisbane
Sue	Manufacturing	Melbourne

Violates 3NF

- Note that there is a functional dependency between *Department* and *Location* (thus transitive dependency).

Third Normal Form

- Problem: redundancy

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

- Solution: split up the relation into two relations:

<u>Employee</u>	Department

<u>Department</u>	Location

Boyce-Codd Normal Form

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

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

- A relation *R* is in **BCNF** if **all** non-trivial FDs that hold over *R* **have** a superkey of R on the LHS. In another words, a relation is in BCNF iff (if and only if) every attribute *is dependent on the key, the whole key and nothing but* the key.
  - Formal: For all *non-trivial* $X \rightarrow A$ *for R*: *X is a superkey for R*
  - *Informal: All dependency arrows come out of candidate keys*

Allowed: AB → C,D

Key: AB

Teacher_id	UnitOfStudy	Address
Mary	COMP9120	One Street
Mary	COMP5313	One Street

Violation: C → B

Violation: Address $\rightarrow$ Teacher-Id

Short 5 mn break:

please stand up, stretch, and move around

Outline

- **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

MultiValued Dependency (MVD)

- Because the 1NF does not allow more than one value for each attribute, and in case there are several of such attributes *that are independent* from each other, there is a need to repeat all combinations of these attributes *so not to infer* a relationship.
- 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:

$X \twoheadrightarrow Y$ (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	$a_1 \ldots a_i$	$a_{i+1} \ldots a_j$	$a_{j+1} \ldots a_n$
t_2	$a_1 \ldots a_i$	$b_{i+1} \ldots b_j$	$b_{j+1} \ldots b_n$
t_3	$a_1 \ldots a_i$	$a_{i+1} \ldots a_j$	$b_{j+1} \ldots b_n$
t_4	$a_1 \ldots a_i$	$b_{i+1} \ldots b_j$	$a_{j+1} \ldots a_n$

- In simple English: if there is a multivalued dependency between X and Y, then this means that *no relationship can be inferred* between Y and Z, i.e., they are *independent* of each other.

MultiValued Dependency (MVD)

Example

Name $\twoheadrightarrow$ *Profession*? *No, this is not an MVD*

The values suggest that whenever

- John is electrician he speaks French.
- John is plumber he speaks Swahili.

NAME	PROFESSION	LANGUAGE
JOHN MARY	ELECTRICIAN DOCTOR	FRENCH SPANISH
JOHN MARY	PLUMBER AUTHOR	SWAHILI CHINESE

- This of course does not make much sense. For instance, John speaks French and Swahili regardless of his multiple professions. Therefore, there should be a multivalued dependency:

  we add a few tuples to reflect this independence.

NAME	PROFESSION	LANGUAGE
JOHN MARY	ELECTRICIAN DOCTOR	FRENCH SPANISH
JOHN MARY	PLUMBER AUTHOR	SWAHILI CHINESE
JOHN MARY	PLUMBER AUTHOR	FRENCH SPANISH
JOHN MARY	ELECTRICIAN DOCTOR	SWAHILI CHINESE

MultiValued Dependency (MVD): An Example

Another example:

UoS ↠ *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 Lecturer. This means that there is an MVD between UoS and Textbook. This also implies that the textbook of a UoS is set independently by the school.

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

- Add one row for each Lecturer of that UoS.

UoS	Textbook	Lecturer
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	**Raghu R**	**Alan F**

MVD properties

MVDs Properties: Armstrong's Axioms

Let X, Y, Z, W be 4 sets of attributes of the relation R:

- Reflexivity: $X \twoheadrightarrow X$
- Complementation: If $X \twoheadrightarrow Y$ Then $X \twoheadrightarrow R - Y$
- Augmentation: If $X \twoheadrightarrow Y$ and $Z \subseteq W$ Then $XW \twoheadrightarrow Y Z$
- Transitivity: If $X \twoheadrightarrow Y$ and $Y \twoheadrightarrow Z$ then $X \twoheadrightarrow Z - Y$
- Replication: If $X \rightarrow Y$, then $X \twoheadrightarrow Y$
- Coalescence: If $X \twoheadrightarrow Y$ and $Z \rightarrow W$ and $Y \cap Z = \emptyset$ and $W \subset Y$, then $X \rightarrow W$

- An MVD $X \twoheadrightarrow Y$ is **trivial** when $Y \subseteq X$ or $X \cup Y = R$

Fourth Normal Form (4NF)

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 lecturers for each textbook that a UoS has listed.

4NF deals with *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 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

Fourth Normal Form (4NF)

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

Employee	Project-id	Personal-phone#
Bob	P1	202-222-2222
Bob	P3	703-333-3333
Bob	P1	703-333-3333
Bob	P3	202-222-2222
Janet	P1	421-534-4452
Nguyen	P7	703-666-6666

Is this relation in 4NF?

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

Employee ↠ Project-id. Note that Employee is *not* a superkey.

Solution: split the above relation into two relations:

Now the two relations are in 4NF!

EMPLOYEE	PROJECT-ID

EMPLOYEE	PERSONAL-PHONE#

Fourth Normal Form (4NF): An Example

Assume the only key to the following relation is the set (UoS, Textbook, Lecturer).

UoS	Textbook	Lecturer
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

Is this relation in 4NF?

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

UoS $\twoheadrightarrow$ Lecturer

and UoS is *not* a superkey.

Solution: Split the above relation into two relations:

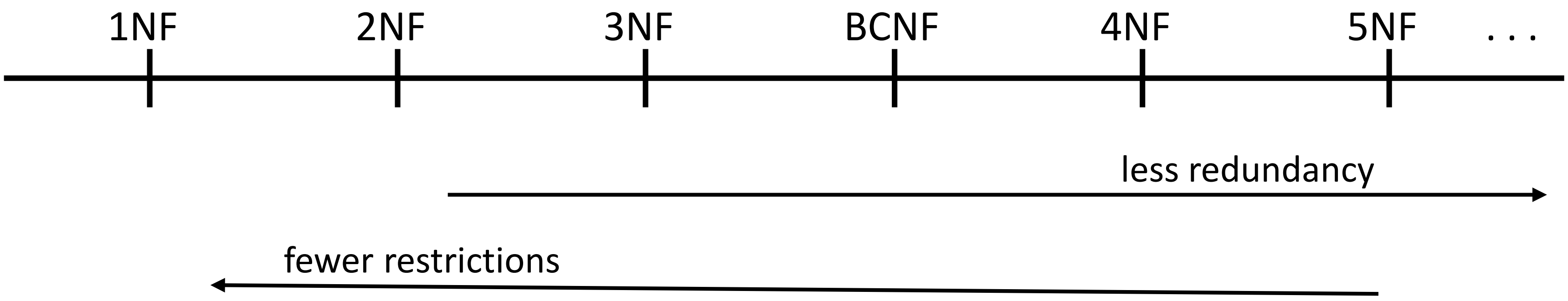
UoS	Lecturer

UoS	Textbook

Now both relations are

In 4NF!

Beyond 4NF



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

Outline

- **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

uosCode → uosName

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

$\bowtie$

Student	UoSCode	Grade
Alice	COMP5138	CR
Alice	COMP5338	D
Bob	COMP5138	P
Clare	COMP5338	HD
David	COMP5338	CR

UoSCode	UoSName
COMP5138	Database Management Systems
COMP5338	Advanced Data Models

Decomposition: Example 2

uosCode → uosName

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

$\bowtie$

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

Schema Decomposition

- 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 no new attributes that do not appear in 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

- Decomposition *properties*:
  - **Dependency preserving:** No FDs are lost in the decomposition
  - **Lossless-join:** Re-joining a decomposition of R should give us back R!

Dependency preservation

› Dependency preservation

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

› Assuming a relaton R is decomposed into relations R_1 R_2 .... R_{n-1} R_n

Idea: 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 $F' = F_1 \cup F_2 \cup ...... \cup F_{n-1} \cup F_n$ 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 $F'^+ = F^+$.

Dependency preservation: An Example

- Assume we have the following relation and FDs:
- R = (A, B, C), F = {A→B, B→C, A→C}

with Key = {A}

This relation is *not* in 3NF because of the transitive FD induced by B → C. We will 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→C}. For R_2, F_2={A→B}. Therefore F$\neq$ {$F_1 \cup F_2$) because we lost the FD: A → 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

› 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

$\bowtie$

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

Lossless join decomposition

- 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.*

Lossless join decomposition

- If R is a relation decomposed into relations R_1, R_2,...., 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:
  - $R = \Pi R_1(R) \infty \Pi R_2(R) \ldots\ldots \infty \Pi R_{n-1}(R) \infty \Pi R_n(R)$ where Π and ∞ 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^+.
  - $R_1 \cap R_2 \rightarrow R_1$
  - $R_1 \cap R_2 \rightarrow R_2$
- 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 either of the resulting relations), then the composition is lossless.

Lossless join decomposition

› BCNF Lossless-Join decomposition algorithm

result = {R}
done = false
compute F^+
while *not done* do
  if there is a relation R_i in result that is not in BCNF then
  {
    let $X \rightarrow Y$ be a nontrivial FD that holds on R_i such that $X \rightarrow R_i$ is not in F^+ and $X \cap Y = \emptyset$
    result= (result - R_i) ∪ (R_i - Y) ∪ (X,Y)
  }
  else
    done = true

If we did not require $X \cap Y = \emptyset$, then those attributes in $X \cap Y$ would not appear in the schema (R_i −Y) and the dependency $X \rightarrow Y$ (across the two relations) would no longer hold.

Decomposing a Schema into BCNF

- 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 = (*customer_id, loan_number, amount*) with *loan_number* → *amount*

but *loan_number* is not a superkey

- Here,
  - X = *loan_number*
  - *Y* = *amount*

So, *loan_info* is replaced by

(X U Y) = (*loan_number, amount*)

(*R* - *Y*) = (*customer_id, loan_number*)

Exercise

- Is the following relation in BCNF? *employee* (*emp_ID*, *name, salary*, *dept_name*, *building_number*) with *dept_name* $\rightarrow$ *building_number*
- If not, give a lossless-join decomposition.
  - Answer: **No**, because *dept_name* is not a superkey
- Here,
  - X = *dept_name*
  - *Y* = *building_number*

So, *employee* is replaced by

- (X U Y) = (*dept_name, building_number*)
- (*R* - *Y*) = (*emp_ID, name, salary, dept_name*)

Exercise

Is the following relation in BCNF? If not, give a lossless-join decomposition

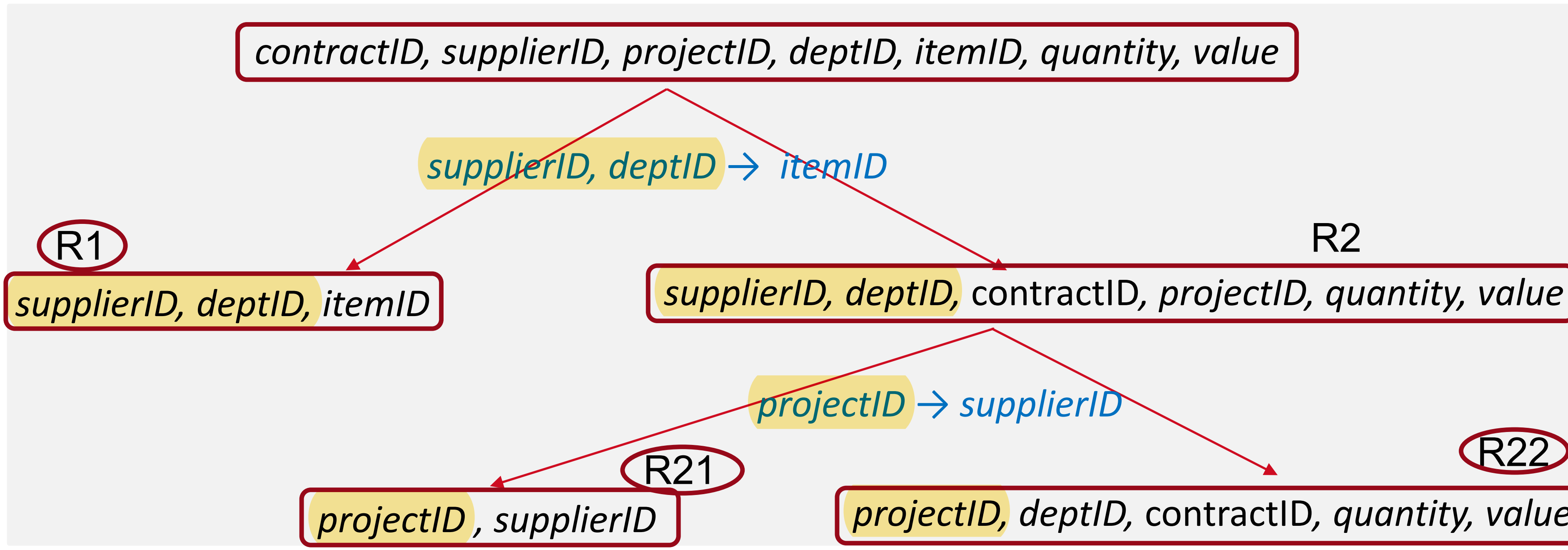
***contracts* (*contractID, supplierID, projectID, deptID, itemID, quantity, value*)**

Functional dependencies are:

contractID → supplierID, projectID, deptID, itemID, quantity, value (***contract ID is key***)

supplierID, deptID → itemID

projectID → supplierID

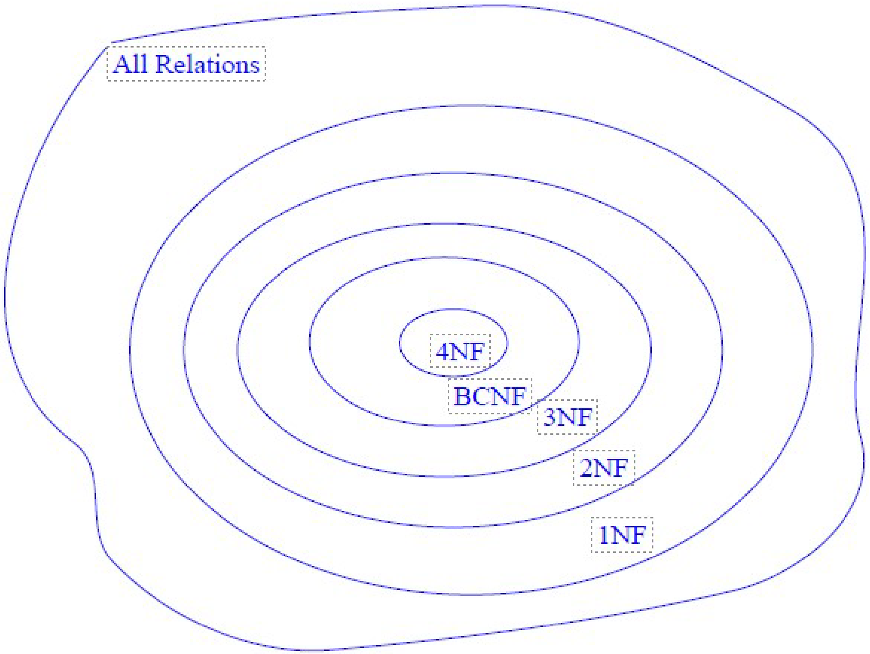


› **Summary**: *Properties of normal forms and their decompositions*

Property	1NF	2NF	3NF	BCNF	4NF
Eliminates redundancy due to FDs	*NO*	*Some*	*Most*	*All*	*All*
Eliminates redundancy due to MVDs	*NO*	*NO*	*NO*	*NO*	*All*
Preserves FDs	*N/A*	*Yes*	*Yes*	*Maybe*	*Maybe*

Relationships of normal forms

Relationship between Normal Forms:



Remember that normalization is

- Geared *towards update*
- 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).

Key Skills

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

Next Week

- Storage and Indexing
  - Storing data in a database
  - Retrieving records from a database
  - B^+ Tree index
- Kifer/Bernstein/Lewis
  - Chapter 9 (9.1-9.5)
- Ramakrishnan/Gehrke
  - Chapter 8
- Ullman/Widom
  - Chapter 8 (8.3 onwards)
- Silberschatz/Korth/Sudarshan (5th ed)
  - Chapter 11 and 12

See you next week!