Functional Dependencies

COMP9311 24T3; Week 4.2

By Wenjie Zhang, UNSW

How good is your DB design?

- **Conceptual Level**
  - How do users interpret the relation schemas and the meaning of their attributes?
- **Physical Level**
  - How the tuples in a base relation are stored and updated?

How good is your DB design?

- **Information Preservation**
  - Does your design correctly capture all attributes, entities and relations?
- **Minimum Redundancy**
  - Does your design minimize redundant storage of the same information and reduce the need for multiple updates?

Example of Redundancy

Suppose we have a table *inst_dept which contains information for both instructor* and *department*.

Result is possible repetition of information, which leads to **update anomalies**.

ID	*name*	*salary*	*dept_name*	*building*	*budget*
22222	Einstein	95000	Physics	Watson	70000
12121	Wu	90000	Finance	Painter	120000
32343	El Said	60000	History	Painter	50000
45565	Katz	75000	Comp. Sci.	Taylor	100000
98345	Kim	80000	Elec. Eng.	Taylor	85000
76766	Crick	72000	Biology	Watson	90000
10101	Srinivasan	65000	Comp. Sci.	Taylor	100000
58583	Califieri	62000	History	Painter	50000
83821	Brandt	92000	Comp. Sci.	Taylor	100000
15151	Mozart	40000	Music	Packard	80000
33456	Gold	87000	Physics	Watson	70000
76543	Singh	80000	Finance	Painter	120000

Update Anomalies

Redundancy in a database means storing a piece of data more than once.

Redundancy is often useful for efficiency and semantic reasons, but creates the potential for consistency problems.

A poor *redundancy control* may cause update anomalies.

Update Anomalies

ID	name	salary	dept_name	building	budget
22222	Einstein	95000	Physics	Watson	70000
12121	Wu	90000	Finance	Painter	120000
32343	El Said	60000	History	Painter	50000
45565	Katz	75000	Comp. Sci.	Taylor	100000
98345	Kim	80000	Elec. Eng.	Taylor	85000
76766	Crick	72000	Biology	Watson	90000
10101	Srinivasan	65000	Comp. Sci.	Taylor	100000
58583	Califieri	62000	History	Painter	50000
83821	Brandt	92000	Comp. Sci.	Taylor	100000
15151	Mozart	40000	Music	Packard	80000
33456	Gold	87000	Physics	Watson	70000
76543	Singh	80000	Finance	Painter	120000

Consider the previous example relation.

- Insertion Anomalies
  - To insert a new employee, we must include the correct values for his/her department or NULLs.
  - How to insert a department with no employees? (set ID to null violates primary key constraint if ID is the primary key)
- Deletion Anomalies
  - What if we delete the last employee in a department? (lose the information of a department)
- Modification Anomalies
  - What if we change the budget of a department? (have to maintain multiple duplication of the same value)

Devise a Theory for what is Good

We want to do two things:

1. Decide whether a particular relation R is in “good” form.
2. If 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

Our theory/properties are defined based on **functional dependencies**.

Attribute Values can be Related

ID	*name*	*salary*	*dept_name*	*building*	*budget*
22222	Einstein	95000	Physics	Watson	70000
12121	Wu	90000	Finance	Painter	120000
32343	El Said	60000	History	Painter	50000
45565	Katz	75000	Comp. Sci.	Taylor	100000
98345	Kim	80000	Elec. Eng.	Taylor	85000
76766	Crick	72000	Biology	Watson	90000
10101	Srinivasan	65000	Comp. Sci.	Taylor	100000
58583	Califieri	62000	History	Painter	50000
83821	Brandt	92000	Comp. Sci.	Taylor	100000
15151	Mozart	40000	Music	Packard	80000
33456	Gold	87000	Physics	Watson	70000
76543	Singh	80000	Finance	Painter	120000

Functional Dependencies

A functional dependency describes a **relation** between attributes

Whenever any two tuples t_1 and t_2 of *r* agree on one attribute α,

they also agree on another attribute β:

$$t_1[\alpha] = t_2[\alpha] \quad \Rightarrow \quad t_1[\beta] = t_2[\beta]$$

This relation is denoted $\alpha \rightarrow \beta$.

Functional Dependencies

ID → Name, Depart_name → Building

ID	*name*	*salary*	*dept_name*	*building*	*budget*
22222	Einstein	95000	Physics	Watson	70000
12121	Wu	90000	Finance	Painter	120000
32343	El Said	60000	History	Painter	50000
45565	Katz	75000	Comp. Sci.	Taylor	100000
98345	Kim	80000	Elec. Eng.	Taylor	85000
76766	Crick	72000	Biology	Watson	90000
10101	Srinivasan	65000	Comp. Sci.	Taylor	100000
58583	Califieri	62000	History	Painter	50000
83821	Brandt	92000	Comp. Sci.	Taylor	100000
15151	Mozart	40000	Music	Packard	80000
33456	Gold	87000	Physics	Watson	70000
76543	Singh	80000	Finance	Painter	120000

Describes the **semantics** or **meaning of the attributes**

Functional Dependencies

The functional dependency

$X \rightarrow Y$ is true (holds)

if and only if

$$t_1[X] = t_2[X] \Rightarrow t_1[Y] = t_2[Y]$$

in relation R

ID	Name	Code	Grade
100	J	3550	A
200	X	3550	B
100	J	4540	B
100	J	4550	A

- Example: R = {ID, Name, Code, Grade}
  - ID → Name (OK)
  - ID → Grade (not OK), ID → Code (not OK)
  - ID, Name → Grade (not OK), ID, Code → Grade (OK)
  - ID, Name → Name (trivial)

Functional Dependencies: Test

Let's see if you understand (Test1)

F: $X \rightarrow Y$

X	Y
a	b
a	?

Functional Dependencies: Test

Let's see if you understand (Test1)

F: $X \rightarrow Y$

X	Y
a	b
a	?

? must be b

Functional Dependencies: Test

Let's see if you understand (Test2)

F: $X \rightarrow Y$

X	Y
a	b
?	b

Functional Dependencies: Test

Let's see if you understand (Test2)

F: X → Y

X	Y
a	b
?	b

? can be any value

Functional Dependencies: Test

Let’s see if you understand (Test3)

F: $X \rightarrow Y$

X	Y	
a	b	
c	b	?

is it okay?

Functional Dependencies: Test

Let's see if you understand (Test3)

F: $X \rightarrow Y$

X	Y	
a	b	
c	b	?

Yes, it doesn't violates $X \rightarrow Y$

Functional Dependencies: Test

Let’s see if you understand (Test4,5)

$X, Y \rightarrow X$?

$X \rightarrow X$?

Functional Dependencies: Test

Let's see if you understand (Test4,5)

$X, Y \rightarrow X$? Yes

$X \rightarrow X$? Yes

Note: Functional dependencies like these are trivial

Functional Dependencies: Test

Let's see if you understand (Test6)

Consider R (A , B) with the following instance r.

1	4
1	5
3	7

On this instance, A → B does NOT hold

FD: relation between two sets

A functional dependency is a relation between two **sets** of attributes.

I.e., the value for a set of attributes determines the value for another set of attributes.

A functional dependency describes the relation between two sets of attributes from a relation.

Examples:

$XY \rightarrow WZ$

$XW \rightarrow Z$

$Z \rightarrow XQ$

Functional Dependencies

A functional dependency is a **constraint** between two sets of attributes for **all** its **relation instances**.

A constraint means a constraint across all it's relation instances (extensions), that it must hold for all relation instances.

F is a set of FD specified on relation R. It must hold on all relation instances.

Constraint on all Relations

Example: *course* → *course_code in* Students

STUDENTS				
id	**course**	**course_code**	**major**	**prof**
1	Database	353	Comp Sci	Smith
2	Chem101	427	Chemistry	Turner
3	Database	353	Comp Sci	Smith

STUDENTS				
id	**course**	**course_code**	**major**	**prof**
1	Database	353	Comp Sci	Yu
4	Agile Dev	821	Comp Sci	Turner
		...		
5	Compiler	237	Comp Sci	Clark

Legal Extensions of R

Relation extensions *r*(*R*) that satisfy the functional dependency constraints are called **legal relation states** (or **legal extensions**) of *R*.

Let *course* → *course_code be the only FD for* Students

STUDENTS				
id	**course**	**course_code**	**major**	**prof**
1	Database	353	Comp Sci	Smith
2	Chem101	427	Chemistry	Turner
3	Database	353	Comp Sci	Clark

Legal

STUDENTS				
id	**course**	**course_code**	**major**	**prof**
1	Database	353	Comp Sci	Yu
4	Agile Dev	821	Comp Sci	Turner
5	Compiler	237	Comp Sci	Clark

Also legal

Notation and Terminology

Let $X \rightarrow Y$ be a functional dependency on relation R

We say that

- $X \rightarrow Y$ *holds* on R

We say that

- X *functionally determines* Y
- Y is *functionally dependent* on X

We say that

- X is *determinant* of the dependency
- Y is *dependent* of the dependency

OR

- X is *left-hand side* of the dependency
- Y is *right-hand side* of the dependency

Functional Dependencies

A *WORKS_ON* relation

- *Ssn* = social security number
- *Pnumber* = project number

Question:
What might be the FDs of *WORKS_ON?*

Ssn, Pnumber → Hours

WORKS_ON

<u>Ssn</u>	<u>Pnumber</u>	Hours
123456789	1	32.5
123456789	2	7.5
666884444	3	40.0
453453453	1	20.0
453453453	2	20.0
333445555	2	10.0
333445555	3	10.0
333445555	10	10.0
333445555	20	10.0
999887777	30	30.0
999887777	10	10.0
987987987	10	35.0
987987987	30	5.0
987654321	30	20.0
987654321	20	15.0
888665555	20	Null

Functional Dependencies

An *EMPLOYEE* relation

- *SSn* = social security number
- *Bdate* = birthday
- *Dnumer* = department number

EMPLOYEE

Ename	Ssn	Bdate	Address	Dnumber
Smith, John B.	123456789	1965-01-09	731 Fondren, Houston, TX	5
Wong, Franklin T.	333445555	1955-12-08	638 Voss, Houston, TX	5
Zelaya, Alicia J.	999887777	1968-07-19	3321 Castle, Spring, TX	4
Wallace, Jennifer S.	987654321	1941-06-20	291Berry, Bellaire, TX	4
Narayan, Ramesh K.	666884444	1962-09-15	975 Fire Oak, Humble, TX	5
English, Joyce A.	453453453	1972-07-31	5631 Rice, Houston, TX	5
Jabbar, Ahmad V.	987987987	1969-03-29	980 Dallas, Houston, TX	4
Borg, James E.	888665555	1937-11-10	450 Stone, Houston, TX	1

Question: What might be the FDs of *EMPLOYEE?*

Ssn → Ename, Address, Bdate

Functional Dependencies

Example: R = {ID, Name, Code, Grade}

r(R) Instance A

ID	Name	Code	Grade
100	J	3550	A
200	X	3550	B
100	J	4540	B
100	J	4550	A

- ID => Name (OK),
- ID => Grade (not OK),
- ID => Code (not OK),
- ID, Name => Grade (not OK),
- ID, Code => Grade (OK).

r(R) Instance B

ID	Name	Code	Grade
100	J	3550	A
200	X	3550	B
100	J	4540	A
100	J	4550	A

- ID => Name (OK)
- ID => Grade (OK),
- ID => Code (not OK)
- ID, Name => Grade (OK),
- ID, Code => Grade (OK).

Important: You can't infer FD's from a relation's instances

Functional Dependencies

Functional dependencies exist to:

- specify the semantics between attributes
  - semantics of a relation should be kept across all its extensions
- specify constraints on a relational schema
- this semantics is not captured by ER

Designing FDs

FD cannot be inferred automatically from a given relation extension r.

So given a relation, where do its FDs come from? Where do we find it?

Deciding the FDs of a table is part of a **design decision**.

- Defined explicitly by someone who knows the semantics of the attributes of R.

Designing FDs

Assume we need to define the FDs of this relation

STUDENTS					
ID	Course	Phone	Major	Prof	Grade

We need to know the semantics of the columns.

Could each ID have a unique phone number and major?

Which Columns are Related?

STUDENTS					
ID	Course	Phone	Major	Prof	Grade

Every ID has a unique phone number and major

- We can say {ID} →{Phone , Major}

Other relations between columns:

- Every course has a unique professor {Course} → {Prof}
- Every ID and course has a unique grade {ID , Course} → {Grade}

Whenever the semantics of two sets of attributes in R indicate that a functional dependency should hold, we specify the dependency as a constraint.

Final Notations

We may denote the attributes sets with/without curly brackets

- With curly brackets, attributes are comma separated
- {X,Y} = XY

The order of the attribute sets doesn't matter

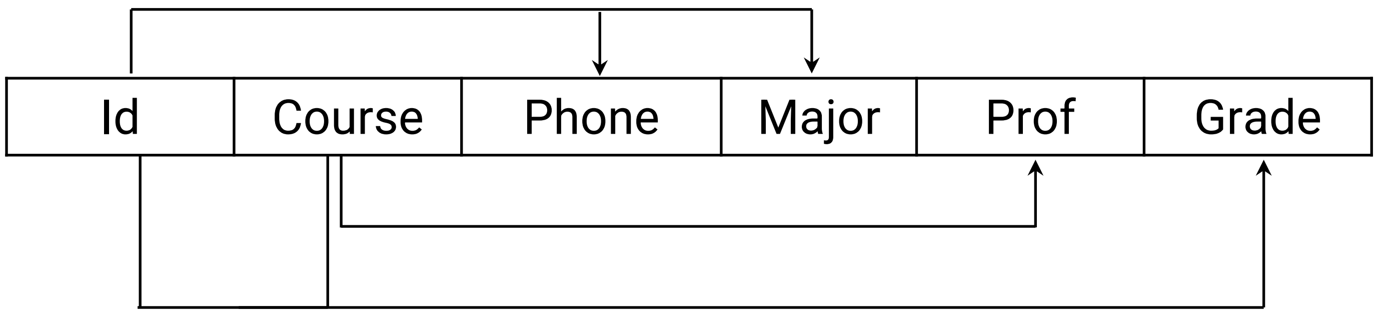
- ZY = YZ
- {Z,Y} = {Y, Z}

Dependency Diagram

Each *horizontal line* represents a FD

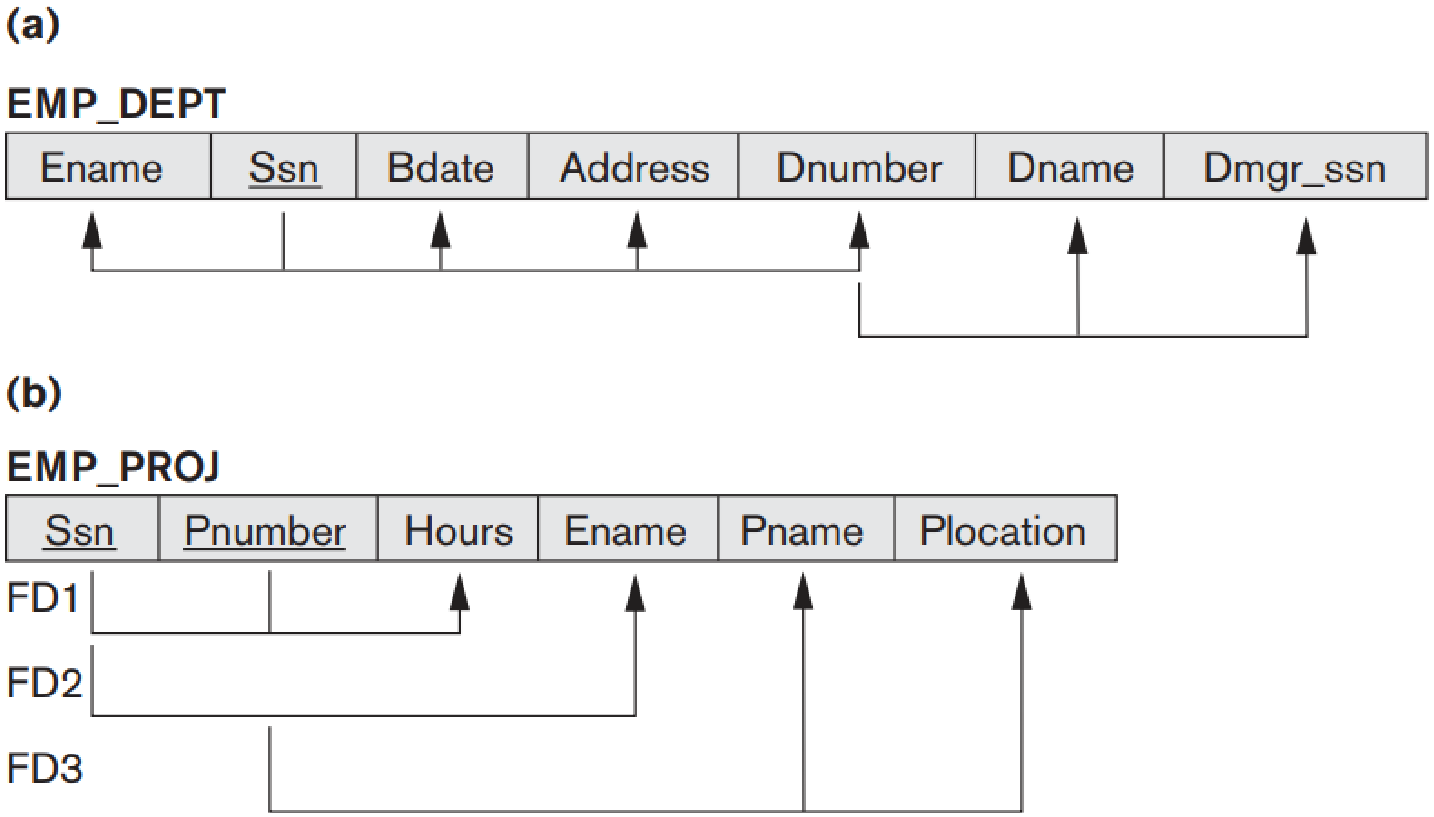
- Left-hand side attr. connected by vertical lines to the line,
- Right-hand side attr. connected by vertical lines <u>with arrows</u>
  - Arrow pointing toward the attributes

Dependency diagram from previous example.



Dependency Diagram (Cont.)

Some more examples of dependency diagrams.



Inferring other FDs

$A \rightarrow B$ and $B \rightarrow C$, what do we know about $A \rightarrow C$?

Given $A \rightarrow B$ and $B \rightarrow C$ on relation R,

We know $A \rightarrow C$ holds on R, given A determines B, and B determines C.

Inferring Other FDs

It's true that given a set F of functional dependencies, there are other functional dependencies that are **logically implied** by F.

$F \models X \rightarrow Y$

Denotes that set of FDs F infers $X \rightarrow Y$ if all relation instances satisfying F also satisfies $X \rightarrow Y$.

Example:

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

$F \models A \rightarrow C$

Usually, the schema designer will only specify the functional dependencies that are semantically obvious.

Armstrong's Axioms

These are the inference rules for functional dependencies

- Rule 1 (reflexivity)
  - if $\beta \subseteq \alpha$, then $\alpha \rightarrow \beta$
- Rule 2 (augmentation)
  - if $\alpha \rightarrow \beta$, then $\gamma\,\alpha \rightarrow \gamma\,\beta$
- Rule 3 (transitivity)
  - if $\alpha \rightarrow \beta$, and $\beta \rightarrow \gamma$, then $\alpha \rightarrow \gamma$
- Where α, β, γ are all (nonempty) sets of attributes

The above are the primary rules/axioms from **Armstrong's Axioms** (1974**)**

Practice

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

These FDs can be inferred/deduced.

$A \rightarrow H$

$AG \rightarrow I$

$CG \rightarrow HI$

(Solutions)

R = (A, B, C, G, H, I)

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

$A \rightarrow H$

by transitivity from $A \rightarrow B$ and $B \rightarrow H$

$AG \rightarrow I$

by augmenting $A \rightarrow C$ to get $AG \rightarrow CG$
then transitivity with given $CG \rightarrow I$

$CG \rightarrow HI$

by augmenting $CG \rightarrow I$ to infer $CG \rightarrow CGI$,
then augmenting $CG \rightarrow H$ to infer $CGI \rightarrow HI$,
followed up by a transitivity

Armstrong's Axioms (Cont.)

Additional Rules we inferred from Armstrong's axioms.

- Rule 4 (**additivity**):
  - If α → β holds and α → γ holds, then α → β γ holds
- Rule 5 (**projectivity**):
  - If α → β γ holds, then α → β holds and α → γ holds
- Rule 6 (**pseudo-transitivity**):
  - If α → β holds and γ β → δ holds, then α γ → δ holds

Proving Secondary Rules

Let's try prove rule 5: projectivity

$\{X \rightarrow Y\,Z\} \models X \rightarrow Y$

Cheat Sheet

F1 (Reflexivity) If $X \supseteq Y$ then $X \rightarrow Y$.

F2 (Augmentation) $\{X \rightarrow Y\} \models XZ \rightarrow YZ$.

F3 (Transitivity) $\{X \rightarrow Y, Y \rightarrow Z\} \models X \rightarrow Z$.

(Solution)

Let's try prove rule 5: <u>projectivity</u>

$\{X \rightarrow Y\,Z\} \models X \rightarrow Y$

Step 1. $X \rightarrow Y\,Z$ (Given)

Step 2. $YZ \rightarrow Y$ (Reflexivity)

Step 3. $X \rightarrow Y$ (Transitivity of 1 and 2)

Cheat Sheet

F1 (Reflexivity) If $X \supseteq Y$ then $X \rightarrow Y$.

F2 (Augmentation) $\{X \rightarrow Y\} \models XZ \rightarrow Y\,Z$.

F3 (Transitivity) $\{X \rightarrow Y,\ Y \rightarrow Z\} \models X \rightarrow Z$.

Proving Secondary Rules

Let's prove rule 6: <u>Pseudo-transitivity</u>

$\{X \to Y \,,\, Y Z \to W\} \models XZ \to W$

Cheat Sheet
F1 (Reflexivity) If $X \supseteq Y$ then $X \to Y$.
F2 (Augmentation) $\{X \to Y\} \models XZ \to Y Z$.
F3 (Transitivity) $\{X \to Y \,,\, Y \to Z\} \models X \to Z$.

(Solution)

Let's prove rule 6: Pseudo-transitivity

$\{X \rightarrow Y, YZ \rightarrow W\} \models XZ \rightarrow W$

Step 1. $X \rightarrow Y$ (Given)

Step 2. $XZ \rightarrow YZ$ (Augmentation of 1)

Step 3. $YZ \rightarrow W$ (Given)

Step 4. $XZ \rightarrow W$ (Transitivity, from 2 and 3)

Cheat Sheet

F1 (Reflexivity) If $X \supseteq Y$ then $X \rightarrow Y$.

F2 (Augmentation) $\{X \rightarrow Y\} \models XZ \rightarrow YZ$.

F3 (Transitivity) $\{X \rightarrow Y, Y \rightarrow Z\} \models X \rightarrow Z$.

Proving Secondary Rules

Let's prove rule 4: Additivity

$\{X \rightarrow Y, X \rightarrow Z\} \models X \rightarrow YZ$

Cheat Sheet

F1 (Reflexivity) If $X \supseteq Y$ then $X \rightarrow Y$.

F2 (Augmentation) $\{X \rightarrow Y\} \models XZ \rightarrow YZ$.

F3 (Transitivity) $\{X \rightarrow Y, Y \rightarrow Z\} \models X \rightarrow Z$.

(Solution)

Let's prove rule 4: Additivity

$\{X \rightarrow Y, X \rightarrow Z\} \models X \rightarrow YZ$

Step 1. $X \rightarrow Y$ (Given)

Step 2 . $XX \rightarrow XY$ (Augmentation of 1); that is, $X \rightarrow XY$

Step 3. $X \rightarrow Z$ (Given)

Step 4. $XY \rightarrow YZ$ (Augmentation of 2)

Step 5. $X \rightarrow YZ$ (Transitivity, from 2 and 4)

Cheat Sheet

F1 (Reflexivity) If $X \supseteq Y$ then $X \rightarrow Y$.

F2 (Augmentation) $\{X \rightarrow Y\} \models XZ \rightarrow YZ$.

F3 (Transitivity) $\{X \rightarrow Y, Y \rightarrow Z\} \models X \rightarrow Z$.

Practice FD Inference

Given F = {A → B, A → C, BC → D}
Prove A → D:

Cheat Sheet
F1 (Reflexivity) If $X \supseteq Y$ then $X \rightarrow Y$.
F2 (Augmentation) $\{X \rightarrow Y\} \models XZ \rightarrow YZ$.
F3 (Transitivity) $\{X \rightarrow Y, Y \rightarrow Z\} \models X \rightarrow Z$.
F4 (Additivity) $\{X \rightarrow Y, X \rightarrow Z\} \models X \rightarrow YZ$.
F5 (Projectivity) $\{X \rightarrow YZ\} \models X \rightarrow Y$.
F6 (Pseudo-transitivity) $\{X \rightarrow Y, YZ \rightarrow W\} \models XZ \rightarrow W$.

(Solution)

Given F = {A → B, A → C, BC → D}
Prove A → D:

Step 1. A → B (Given)

Step 2. A → C (Given)

Step 3. A → BC (Additivity, from 1 and 2)

Step 4. BC → D (Given)

Step 5. A → D (Transitivity, from 3 and 4)

Cheat Sheet
F1 (Reflexivity) If $X \supseteq Y$ then $X \rightarrow Y$.
F2 (Augmentation) $\{X \rightarrow Y\} \models XZ \rightarrow YZ$.
F3 (Transitivity) $\{X \rightarrow Y, Y \rightarrow Z\} \models X \rightarrow Z$.
F4 (Additivity) $\{X \rightarrow Y, X \rightarrow Z\} \models X \rightarrow YZ$.
F5 (Projectivity) $\{X \rightarrow YZ\} \models X \rightarrow Y$.
F6 (Pseudo-transitivity) $\{X \rightarrow Y, YZ \rightarrow W\} \models XZ \rightarrow W$.

Closure of F

Definition. the set of all dependencies that can be inferred from F is called the **closure** of F.

F+ denotes the closure of F.

F+ includes dependencies in F.

Note: We typically reserve F to denote the set of functional dependencies that are specified on relation schema R.

The Procedure for Computing F+

To compute the closure of a set of functional dependencies F:

```
F+ = F
repeat
    for each functional dependency f in F+
        apply reflexivity and augmentation rules on f
        add the resulting functional dependencies to F+
    for each pair of functional dependencies f1 and f2 in F+
        if f1 and f2 can be combined using transitivity
            then add the resulting functional dependency to F+
until F+ does not change any further
```

The Procedure for Computing F+

F = { X → Y, Y → Z}

F+ = {XY → X, XY → Y, XY → Z, XZ → X, XZ → Y,
XZ → Z, XYZ → X, XYZ → Y, XYZ → Z, XY → XY,
XY → YZ, XY → XZ, …}

Checking Membership by F+

Given F = { $X \rightarrow Y$, $Y \rightarrow Z$}

Question: Can $X \rightarrow Z$ be inferred or derived from the FDs in F?

How to do it? Check $X \rightarrow Z$ by computing F+?

Checking Membership by F+

Given F = { X → Y, Y → Z}

Question: Can X → Z be inferred or derived from the FDs in F?

How to do it? Check X → Z by computing F+?

F+ = {XY → X, XY → Y, XY → Z, XZ → X, XZ → Y,
XZ → Z, XYZ → X, XYZ → Y, XYZ → Z, XY → XY,
XY → YZ, XY → XZ, … , X → Z , … }

Oh yes… X → Z is in the closure of F.

Checking Membership by F+

Given F = { X → Y, Y → Z}

Question: Can X → Z be inferred or derived from the FDs in F?

How to do it? Check X → Z by computing F+?

F+ = {XY → X, XY → Y, XY → Z, XZ → X, XZ → Y, XZ → Z, XYZ → X, XYZ → Y, XYZ → Z, XY → XY, XY → YZ, XY → XZ, … , X → Z , … }

Oh yes… X → Z is in the closure of F.

Problem: In real life, it is impossible to specify all possible functional dependencies for a given situation. The size of F+ is always **exponential** size w.r.t |F|.

Closure of Attributes

Given F = { $X \rightarrow Y$, $Y \rightarrow Z$}

Question: How else to check if $X \rightarrow Z$ without computing F+ ?

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

Realistically:
Narrow our attention to X, which is smaller than F.
Compute X+ instead of F+
Then check if Z is covered by X+

X+ is the **largest** set of attributes functionally determined by X.

Closure of Attribute Sets

Pseudocode to the closure of α under *F*

```
result := a;
while (changes to result) do
        for each β → γ in F do
                begin
                        if β ⊆ result then  result := result ∪ γ
                end
```

When no additional changes to result is possible, the final value of variable result is α^+

Algorithm to Compute X^+

An **algorithm** for you to follow step by step

```
X+ := X;
change := true;
while change do
        begin
        change := false;
        for each FD W → Z in F do
                begin
                if (W ⊆ X+) and (Z – X+ ≠ ∅) then do
                        begin
                        X+ := X+ ∪ Z;
                        change := true;
                        end
                end
        end
```

Exercise

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

Practice: **Compute A+**

Cheat Sheet:

```
X⁺ := X;
change := true;
while change do
        begin
        change := false;
        for each FD W → Z in F do
          begin
          if (W ⊆ X⁺) and (Z – X⁺ ≠ ∅)
          then do
                begin
                X⁺ := X⁺ ∪ Z;
                change := true;
                end
          end
        end
```

(Solution)

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

Task: **Compute {A}+**

1st scan of F:

X+ := {A}
X+ := {A, B}
X+ := {A, B, C}

2nd scan of F:

X+ := {A, B, C, D }

3rd scan of F: no change, therefore, the algorithm terminates.

{A}+ := {A, B, C, D }

Cheat Sheet:

```
X⁺ := X;
change := true;
while change do
        begin
        change := false;
        for each FD W → Z in F do
           begin
           if (W ⊆ X⁺) and (Z − X⁺ ≠ ∅)
           then do
                  begin
                  X⁺ := X⁺ ∪ Z;
                  change := true;
                  end
           end
        end
```

Recall of Attribute Set Closure

$R = (A, B, C, G, H, I)$

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

We know $(AG)^+ = ABCGHI$

Observation: could AG be a candidate key?

- **Is AG a super key?**
  - Does $AG \rightarrow R$? => Is $(AG)^+ = R$?
- **Is any subset of AG a super key?**
  - Does $A \rightarrow R$? => Is $(A)^+ = R$?
  - Does $G \rightarrow R$? => Is $(G)^+ = R$?

Functional Dependencies (Cont.)

K is a super key for relation schema R if and only if $K \rightarrow R$

K is a candidate key for R if and only if

- $K \rightarrow R$, and
- for no $\alpha \subset K$, $\alpha \rightarrow R$

Answer

$R = (A, B, C, G, H, I)$

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

We know $(AG)+ = ABCGHI$

Observation: could AG be a candidate key?

Is AG a super key?

Does $AG \rightarrow R$? => Is $(AG)^+ = R$? Yes, so AG is a super key

Is any subset of AG a super key?

Does $A \rightarrow R$? => Is $(A)^+ = R$? No

Does $G \rightarrow R$? => Is $(G)^+ = R$? No

So AG is a candidate key

Procedurally Determine Keys

How to compute a candidate key of a relation R based on the FD's belonging to R

Algorithm:

- *Step 1 : Assign a super-key of R in F to X.*
- *Step 2 : Iteratively remove attributes from X while retaining the property X+ = R till no reduction on X is possible.*
- *The remaining X is a key.*

Let's try an example

Practice

Given:

$R = \{A, B, C, D\}$

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

Step 1 : Assign a super-key of R in F to X.

Step 2 : Iteratively remove attributes from X while retaining the property $X^+ = R$ till no reduction on X is possible.

The remaining X is a key.

(Solution)

Given:
R = {A, B, C, D}
F = { A → B, BC → D, A → C }

Let X = {A, B, C} ({A, B, C, D} is also a super key)

A cannot be removed because {BC}+ = {B, C, D} ≠ R

B can be removed because {AC}+ = {A, B, C, D} = R

We remove B from X and update X to be { A, C}

C can be further removed because {A}+ = {A, B, C, D}

We remove C from X and update X to be { A}

Step 1 : Assign a super-key of R in F to X.
Step 2 : Iteratively remove attributes from X while retaining the property $X^+ = R$ till no reduction on X is possible.
The remaining X is a key.

Compute all Candidate Keys

Given a relational schema R and a set of functional dependencies F on R, find all the possible ways we can identify a row.

Note: we know how to compute one candidate key already.

Compute All the Candidate Keys

Given a relational schema R and a set F of functional dependencies on R, the algorithm to compute all the candidate keys is as follows:

T := ∅

Main:

    X := S where S is a super key which does not contain any candidate key in T

    remove := true

    While remove do

        For each attribute A ∈ X

        Compute $\{X-A\}^+$ with respect to F

        If $\{X-A\}^+$ contains all attributes of R then

            X := X – {A}

        Else

            remove := false

    T :=T ∪ X

Repeat *Main* until no available S can be found. Finally, T contains all the candidate keys.

Compute all Candidate Keys

Given relation R(A, B, C, D, E)

with set of FDs {$A \rightarrow B$, $BC \rightarrow A$, $D \rightarrow E$}

Find all the candidate keys for relation R

(Solution)

Step 1:

Let X := {A, B, C, D}

Step 2:

Try to remove A

{B, C, D}+ = {A, B, C, D, E}

Thus X := {B, C, D}

Steps 3,4,5:

Attempts to remove B, C, D separately

{C, D}+ = {C, D, E}

{B, D}+ = {B, D, E}

{B, C}+ = {A, B, C}

None can be removed

So {<u>B, C, D</u>} is a candidate key and add to T

(Solution)

Step 6:
Find another super key
Let X := {A, C, D}

Step 7,8,9:
Attempts to remove A, C, D separately

{C, D}+ = {C, D, E}

{A, D}+ = {A, B, D, E}

{A, C}+ = {A, B, C}

None cannot be removed
So, <u>{A, C, D} </u>is another candidate key and add to T

(Solution)

Step 10:
Cannot find any other super keys,

Conclusion: candidate keys are *{B, C, D}* and {A, C, D}

Lecture Learning Outcomes

Take aways

- Functional Dependencies
- Armstrong’s axioms
- Given a FD, check if the FD can be derived from a given set of FD
- How to compute one candidate key
- How to compute all candidate keys