CSI2532 - Tutorial 9 - Normalization

Overview

The intent of today's tutorial is to give you hands up experience with functional dependencies (FD) and normalization.

Dependency rules

Reflexive rule	Augmentation rule	Transitivity rule
if $\beta \subseteq \alpha$, then $\alpha \to \beta$	if $\alpha \to \beta$, then $\gamma\alpha \to \gamma\beta$	if $\alpha \to \beta$ and $\beta \to \gamma$, then $\alpha \to \gamma$

Derived rules

Union rule	Decomposition rule	Pseudo-transitivity rule
if $\alpha \to \beta$, $\alpha \to \gamma$, then $\alpha \to \beta\gamma$	if $\alpha \to \beta\gamma$, then $\alpha \to \beta$ and $\alpha \to \gamma$	if $\alpha \to \beta$ and $\gamma\beta \to \delta$, then $\alpha\gamma \to \delta$

Use the rules above to prove a dependency holds. Or find a counter example here there is ambiguity so that α could not uniquely identify β.

Reflexive rule
if $\beta \subseteq \alpha$,
then $\alpha \to \beta$

Augmentation rule
if $\alpha \to \beta$,
then $\gamma\alpha \to \gamma\beta$

Transitivity rule
if $\alpha \to \beta$ and $\beta \to \gamma$
then $\alpha \to \gamma$

Union rule
if $\alpha \to \beta$, $\alpha \to \gamma$
then $\alpha \to \beta\gamma$

Decomposition rule
if $\alpha \to \beta\gamma$
then $\alpha \to \beta$ and $\alpha \to \gamma$

Pseudo-transitivity rule
if $\alpha \to \beta$ and $\gamma\beta \to \delta$
then $\alpha\gamma \to \delta$

Super Key

A super key in one where K → R. To prove that you need to show K's closure K+ includes all relations (K+ = R).

Test superkey K

Test α = K

Check α+ → R

Candidate Key

A candidate key is a minimized super key. So there should be no $\alpha \subset K$ where $\alpha+ = R$.

Test α candidate K

Test $\alpha + \rightarrow K$

$\exists\ \beta \subset \alpha$ test ! $\beta + \rightarrow R$

α size n, and β size n-1

α+ (Closure of attribute set)

To calculate $\alpha+$ run through all functional dependencies $\beta \rightarrow \gamma$, if $\alpha \subseteq \beta$ then add γ to a+

calc... α^+

```
α+ := α
do {
  foreach (β→γ in F) {
    if (β⊆α+) {
      α+ ∪ γ
    }
  }
} while (changes to α+);
```

F+ (closure of all functional dependencies)

Here is the original algorithm we learned for calculating F+.

calc… F+

```
F+ := F
do {
  foreach (f in F+) {
    F+ ∪ apply(reflexivity, f);
    F+ ∪ apply(augmentation, f);
  }
  foreach (f1, f2 in F+) {
    F+ ∪ apply(transitivity, f1, f2);
  }
} while (changes to F+);
```

But, we can instead use α+. Calculate the α+ for every combination. Create all combinations of X → Y.

The official algorithm is

```
∃ γ⊆R find γ+
∃ S⊆γ+ output FD γ⊆
```

Compute F+

R=(A, B, C)

F={A→B, B→C}

∅+ = ∅ so ∅→∅

(A)+ = A
= AB
= ABC

(B)+ = B
= BC

(C)+ = C

(AB)+ = AB
= ABC

(AC)+ = AC
= ABC

(BC)+ = BC

(ABC)+ = ABC

Compute F+

R=(A, B, C)

F={A→B, B→C}

∅→∅

A→∅, A→A, A→B, A→C, A→AB, A→AC, A→BC, A→ABC

B→∅, B→B, B→C, B→BC

C→∅, C→C

AB→∅, AB→A, AB→B, AB→C, AB→AB, AB→AC, AB→BC, AB→ABC

AC→∅, AC→A, AC→B, AC→C, AC→AB, AC→AC, AC→BC, AC→ABC

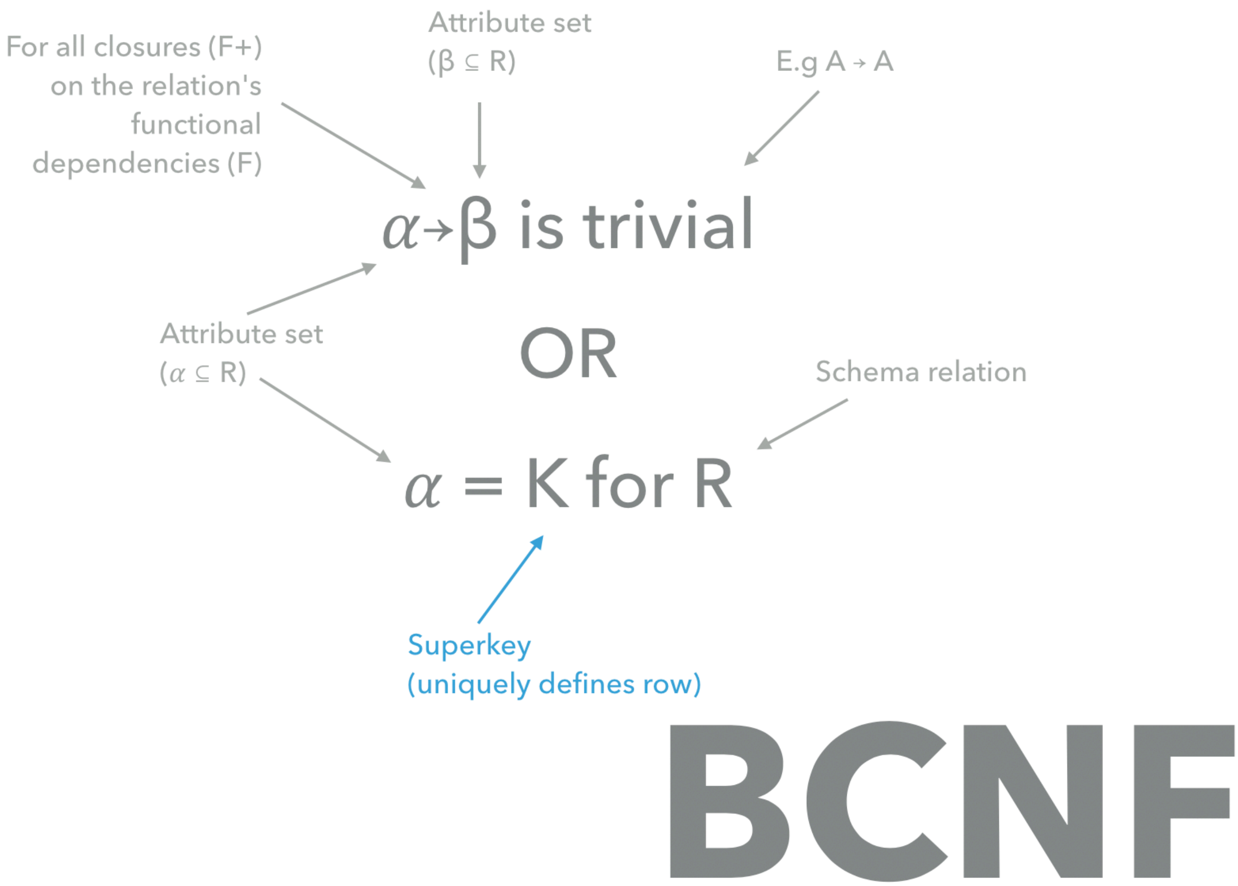
BC→∅, BC→B, BC→C, BC→BC

ABC→∅, ABC→A, ABC→B, ABC→C, ABC→AB, ABC→AC, ABC→BC, ABC→ABC

BCNF Test

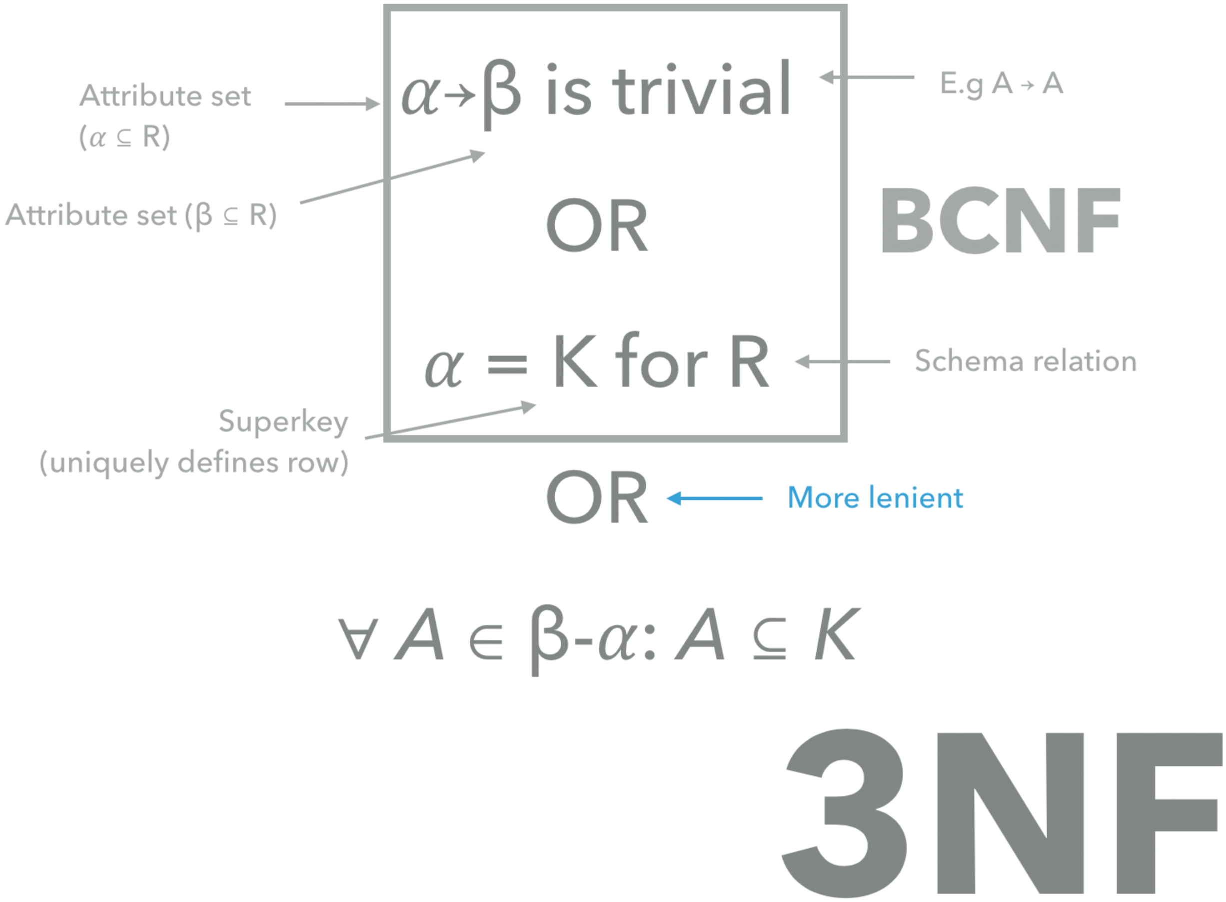
To test BCNF, you need to look all closures of F (F+). If you find a relation $\alpha \rightarrow \beta$ where $\alpha \not\subset R$

then the relation is NOT BCNF. The official rule is `α = K for R`



3NF Test

If not BCNF, then if you a relation $\alpha \to \beta$ where an attribute in β (but not α, aka β - α) is not in K (any candidate key) then it is not 3NF. The official rule is `∀ A ∈ β−α: A ⊆ K`



Canonical Cover

Continue until Fc does not change.

1. Apply the union rule
2. Remove any extraneous attribute when looking at all $\alpha \to \beta$

Canonical Cover

```
Fc := F
do {
 foreach (f1, f2 in Fc) {
  // α1→β1 and α1→β2
  // into α1→β1β2
  reduce(union, f1, f2);
 }
 foreach (α→β in Fc) {
  if with Fc find extraneous A in α or β {
   delete(A, α→β)
  }
 }
} while (changes to Fc);
```

Based on extraneous tests

Testing for extraneous A in $\alpha \to \beta$

Remove "left"

$\gamma = \alpha - \{A\}$

Check if $\gamma \to \beta$
(Check $\beta \subseteq \gamma+$)

Remove "right"

$F' = (F - \{\alpha \to \beta\}) \cup \{\alpha \to (\beta - A)\}$

Check $A \in \alpha+$ under F'

Extraneous A in α

After you remove the attribute, try to provide the *stronger* functional dependency still exists.

Consider

```
R=(A, B, C, D)
F={
  AB→C,
  A→D,
  D→C
}
```

Can we (safely) remove “B” in AB → C?

We need to prove that A → C can be derived from the other functional dependencies.

YES we can (using the rules). As A→D and D→C so A→C. So "B" is extraneous. OR YES (calculating using α+ using F). (A)+ = ADC so yes A→C.

Extraneous A in β

After you remove the attribute, try to show you can *recreate* the stronger functional dependency based on the now weaker one.

Consider

```
R=(A, B, C, D)
F={
  AB→CD,
  A→C
}
```

Can we (safely) remove “C” in AB→CD?

We need to prove that using only AB→D (a weaker claim) can we get back to our stronger claim of AB→CD.

YES we can (using rules). Using A→C we augment it to AB→C, and then union that with the (weaker) AB→D we get back to AB→CD.

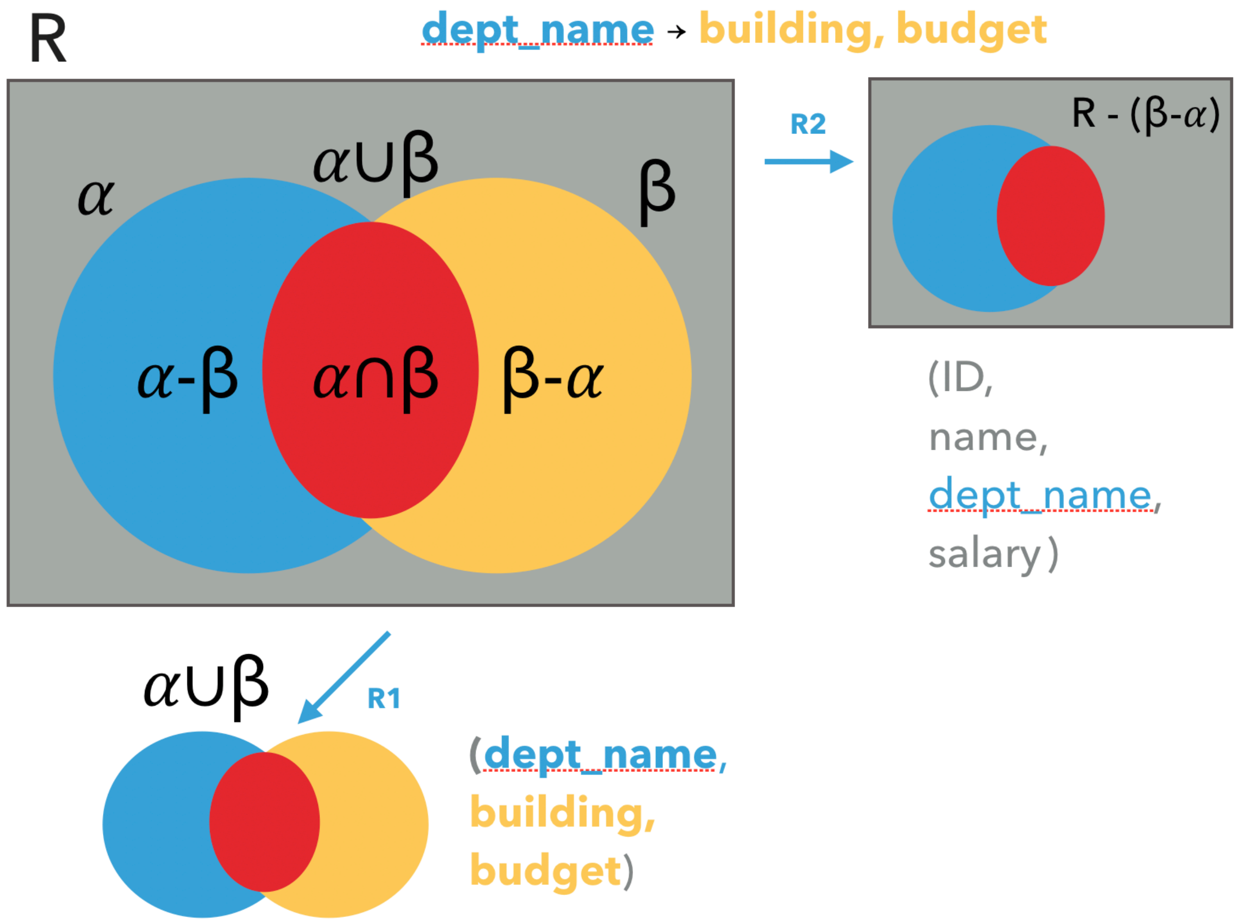
OR YES (calculating β+ using F'). Compute (AB)+=ABCD from F' = {AB→D, A→C} which contains AB→CD so "C" is extraneous.

BCNF Decomposition

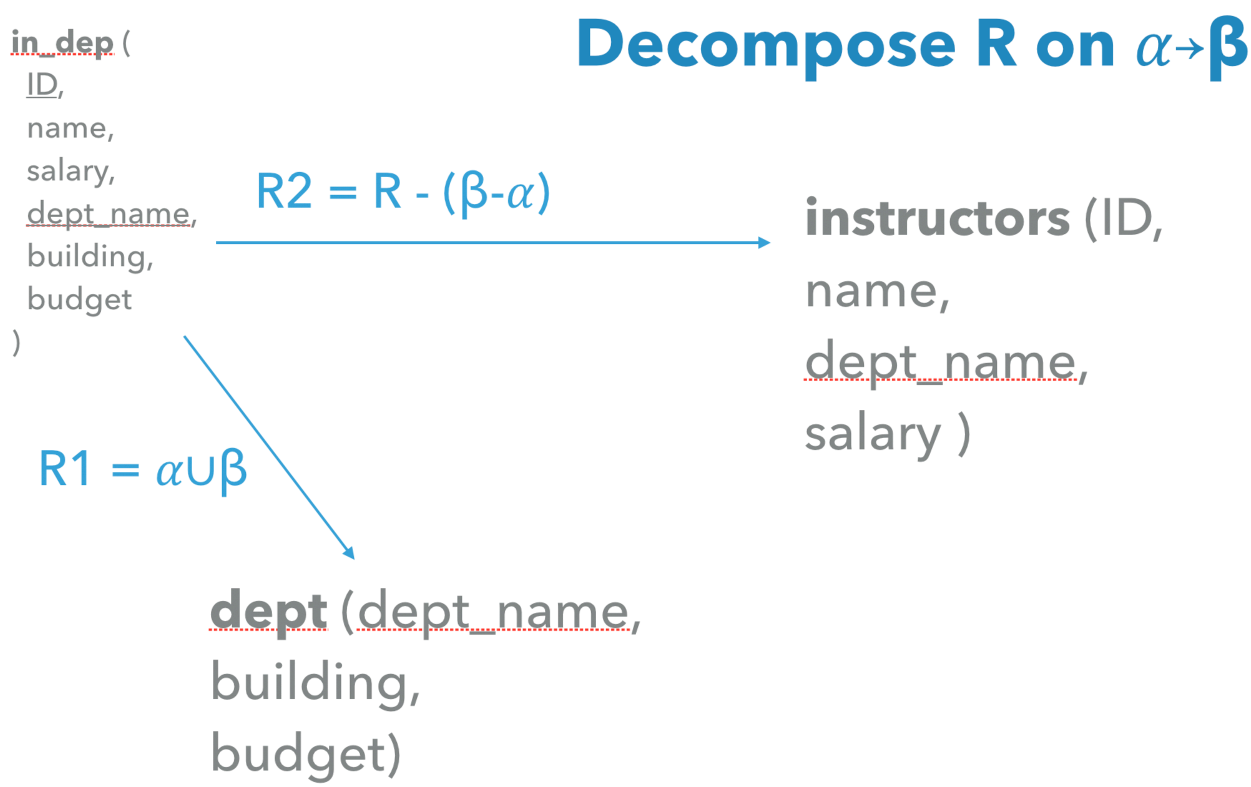
result := $\{R\}$;
done := false;
compute F^+;
while (not *done*) **do**
    if (there is a schema R_i in *result* that is not in BCNF)
        then begin
            let $\alpha \rightarrow \beta$ be a nontrivial functional dependency that holds on R_i such that $\alpha \rightarrow R_i$ is not in F^+,
                and $\alpha \cap \beta = \emptyset$;
            result := (*result* – R_i) $\cup$ ($R_i - \beta$) $\cup$ (α, β);
        end
        else *done* := **true;**

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

For the relation that fails BCNF


R
dept_name → building, budget
α
α∪β
β
α-β
α∩β
β-α
R2
R - (β-α)
(ID,
name,
dept_name,
salary)
α∪β
R1
(dept_name,
building,
budget)

For example,



Q1a: Testing Normal Forms

Consider the following relation and functional dependencies.

```
R=(A,B,C,D)
F={
  AB → C,
  C → D,
  D → A
}
```

a. List all candidate keys of R.

b. Is R in 3NF? BCNF?

Solution

a) Calculate candidate keys

Let's find all super keys by calculating α+.

Start with all n=1 keys

α	α+	Candidate key?
A	A	
B	B	
C	CDA	
D	DA	

No keys yet, so let's test all n=2 keys

α	α+	Candidate key?
AB	ABCD	yes
AC	ACD	
AD	AD	
BC	BCDA	yes
BD	BDAC	yes
CD	CDA	

Lets test all n=3 keys. Write out all combinations to avoid missing one by accident, but we can ignore any with AB, BC and CD.

α	α+	Superkey?
ABC	N/A	Ignore due to "AB"
ABD	N/A	Ignore due to "AB"
ACD	N/A	Ignore due to "CD"
BCD	N/A	Ignore due to "BC"

We can also ignore n=4, as it's not a minimized key.

The candidate keys are

- AB
- BC
- BD

b1) Test for BCNF

Before we calculate F+, let's look at F to find any that break BCNF.

We have a functional dependency C → D, but C is not equal to {AB, BC, BD}, same for D → A. So R is not BCNF.

b2) Test for 3NF

Same as above, but now testing our additional 3NF rule.

For AB → C we see that C ⊆ BC For C → D we see that D ⊆ BD For D → A we see that A ⊆ AB

For a complete test we should look at all of F+, but as we know that every attribute in α is part of a candidate key, so we know that YES this relation is 3NF.

Q1b: Testing Normal Forms

Consider the following relation and functional dependencies.

```
R=(A,B,C,D)
F={
  A → B,
  B → C,
  C → D,
  D → A
}
```

a. List all candidate keys of R.

b. Is R in 3NF? BCNF?

Solution

a) Calculate candidate keys

Let's find all super keys by calculating α+.

Start with all n=1 keys

α	$\alpha+$	Candidate key?
A	ABCD	yes
B	BCDA	yes
C	CDAB	yes
D	DABC	yes

b) Test for BCNF

All attributes are candidate keys, so we can calculating as any n+1 would include an attribute that is already a superkey. CNF. Any BCNF is also 3NF, so yes it is 3NF.

Q1c: Testing Normal Forms

Consider the following relation and functional dependencies.

```
S=(A,B,C,D)
F={
  B → C,
  C → A,
  C → D
}
```

a. List all candidate keys of R.

b. Is R in 3NF? BCNF?

Solution

a) Calculate candidate keys

The candidate key is

- B

b) Test for BCNF / 3NF

Not BCNF, not 3NF

Q1d: Testing Normal Forms

Consider the following relation and functional dependencies.

```
R=(A,B,C,D)
F={
  ABC → D,
  D → A
}
```

a. List all candidate keys of R.

b. Is R in 3NF? BCNF?

Solution

a) Calculate candidate keys

The candidate keys are

- ABC
- BCD

b) Test for BCNF / 3NF

NO to BCNF, YES to 3NF

Q1e: Testing Normal Forms

Consider the following relation and functional dependencies.

```
R=(A,B,C,D)
F={
  A → C,
  B → D
}
```

a. List all candidate keys of R.

b. Is R in 3NF? BCNF?

Solution

a) Calculate candidate keys

The candidate keys are

- AB

b) Test for BCNF / 3NF

Not BCNF, not 3NF

Q2a: Testing Functional Dependency

Consider the following relation and functional dependencies.

```
R=(A,B,C,D,E,F)
F={
  AB → C,
  BC → AD,
  D → E,
  CF → B
}
```

Does AB → D hold? If so, show a formal proof; otherwise, give a counterexample.

Solution

Yes, AB → D holds.

```
AB → B (reflexivity)
AB → BC (union of AB → B and AB → C)
AB → AD (transitive AB → BC and BC → AD)
AB → D (decomposition into AB -> A and AB -> D)
```

OU

```
(AB)+ = ABCDE
AB -> D
```

Q2b: Testing Functional Dependency

Consider the following relation and functional dependencies.

```
R=(A,B,C)
F={
  AB → C
}
```

Does A → C hold? If so, show a formal proof; otherwise, give a counterexample.

Solution

No.

Here we see that YES AB → C, but ! A → C as we have 1 can return 98 OR 99.

A	B	C
1	1	98
1	2	99

Q2c: Testing Functional Dependency

Consider the following relation and functional dependencies.

```
R=(A,B,C)
F={
  AB → C
}
```

Does B → C hold? If so, show a formal proof; otherwise, give a counterexample.

Solution

No.

Here we see that YES AB → C, but ! A → C as we have 1 can return 98 OR 99.

A	B	C
1	1	98
2	1	99

Q2d: Testing Functional Dependency

Consider the following relation and functional dependencies.

```
R=(A,B,C)
F={
  AB → C
}
```

Does A → C OR B → C hold? If so, show a formal proof; otherwise, give a counterexample.

Solution

No.

Here we see that YES AB → C, but

- A → C does not hold as A = 2 returns C = 98 or 99
- B → C does not hold as B = 1 returns C = 97 or 98

A	B	C
1	1	97
2	1	98
2	2	99

Q3: Canonical Cover

Compute a canonical cover for

```
F={
  B → A,
  D → A,
  AB → D
}
```

Solution

```
Fc = {  B → A,  D → A, AB → D }
     --- (loop)
     =  NO CHANGE from union
       > Check "B" extraneous in AB → D (so show A → D in F)... (A)+ = A, so
       > Check "A" extraneous in AB → D  (so show B → D in F).... (B)+ = BAD
     = { B → A,  D → A, B → D }
     --- (loop)
     = { B → AD,  D → A }  APPLY UNION
      > Check "D" extraneous in B → AD.  Check (B)+ in {B → A, D → A}, (B)+
      > Check "A" extraneous in B → AD.  Check (B)+ in {B → D, D → A}, (B)+
     = { B → D,  D → A }
     --- (loop)
     =  NO CHANGE from union
     =  No extraneous attributes left
     = { B → D,  D → A }
```

Q4: BCNF Decomposition

Produce a BCNF decomposition of R.

```
R = ABCDEFGH
F = {
  ABH → C,
  A → DE,
  BGH → F,
  F → ADH,
  BH → GE
}
```

Solution

For BCNF you need only find a FD where α != K

Let us start with n=1, just check those FDs with one attribute.

α	α+	Candidate key?
A	ADE	no
F	FADH	no

So A → DE breaks BCNF.

```
R1 = (A ∪ DE) = ADE
R2 = (ABCDEFGH - DE) = ABCFGH
```

Let's decompose into F1, F2. Remove all attributes not in the new relations, make sure those FDs are valid decompositions, if not, then remove them.

```
F1 = { A → ∅ ✓,  A → DE ✓,  ∅ → ∅ ✓,  ∅ → AD x,  ∅ → E x} = { A → DE }
F2 = { ABH → C ✓, A → ∅ ✓,  BGH → F ✓,  F → AH ?,  BH → G ?} = { ABH → C,
 BGH → F,  F → AH,  BH → G }
```

We know F → AH based on decomposition (F → ADH implies F → AH). We also know BH → G based on decomposition (BH → GE implies BH → G).

Now R1 is BCNF as α = A = K.

Let's retest R2 based on the decomposed FDs

α	α+	Candidate key?
F	FAH	no

So F → AH breaks BCNF in R2. Let's decompose R2.

```
R2a = (F ∪ AH) = AFH
R2b = (ABCFGH - AH) = BCFG
```

Let's decompose into F2a, F2b.

```
F2a = { AH → ∅ ✓,  H → F ?,  F → AH ✓,  H → G ? } = { F → AH }
F2b = { B → C ?,  B → F ?,  F → ∅ ✓,  B → G ? } = {}
```

Let's us consider (B)+ = B, so none of the B → implications hold. Both R2a and R2b are BCNF so we are done.

```
R1 = ADE
R2a = FAH
R2b = BCFG
```