NUS | Computing
National University of Singapore

CS2102 Database Systems

Lecture 11 – Boyce-Codd Normal Form

Roadmap

Refinement

3NF BCNF

Normal Forms

Triggers

Functional Dependencies

Stored Procedures

Query

Relational Algebra

SQL Query

Design

SQL DDL

ER Diagram

Constraints

Roadmap

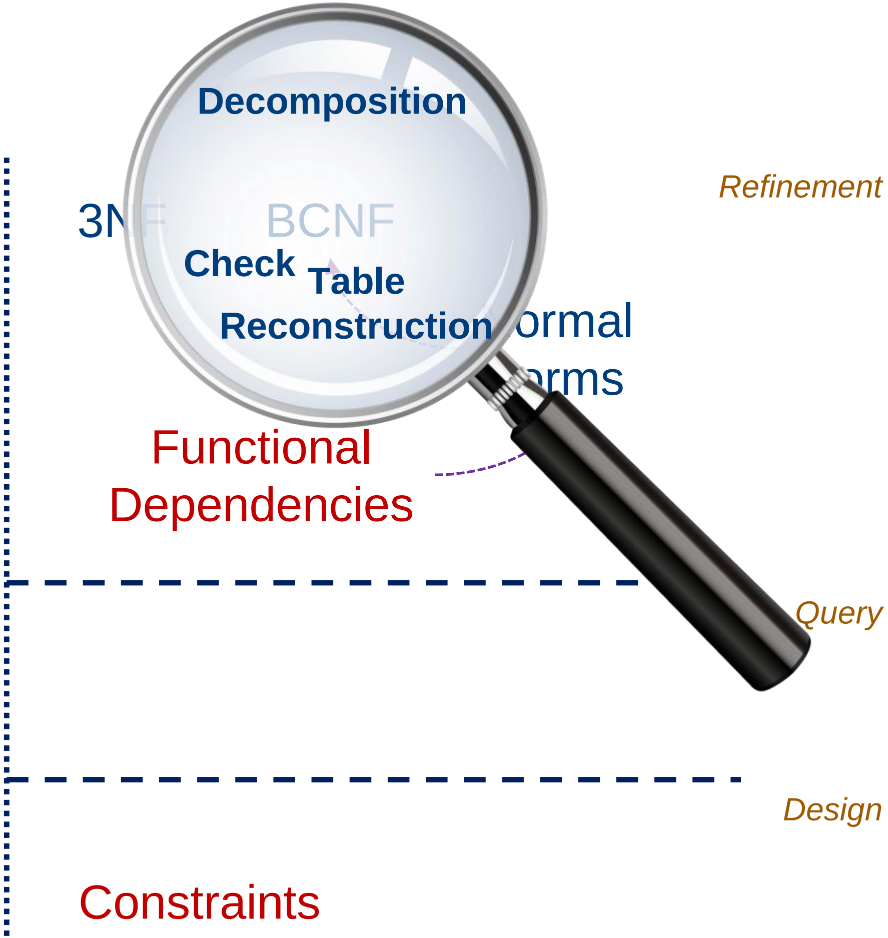
- We will do this step by step
  - Recap FD *(with non-trivial & decomposed FD)*

    ⇩
  - Define BCNF

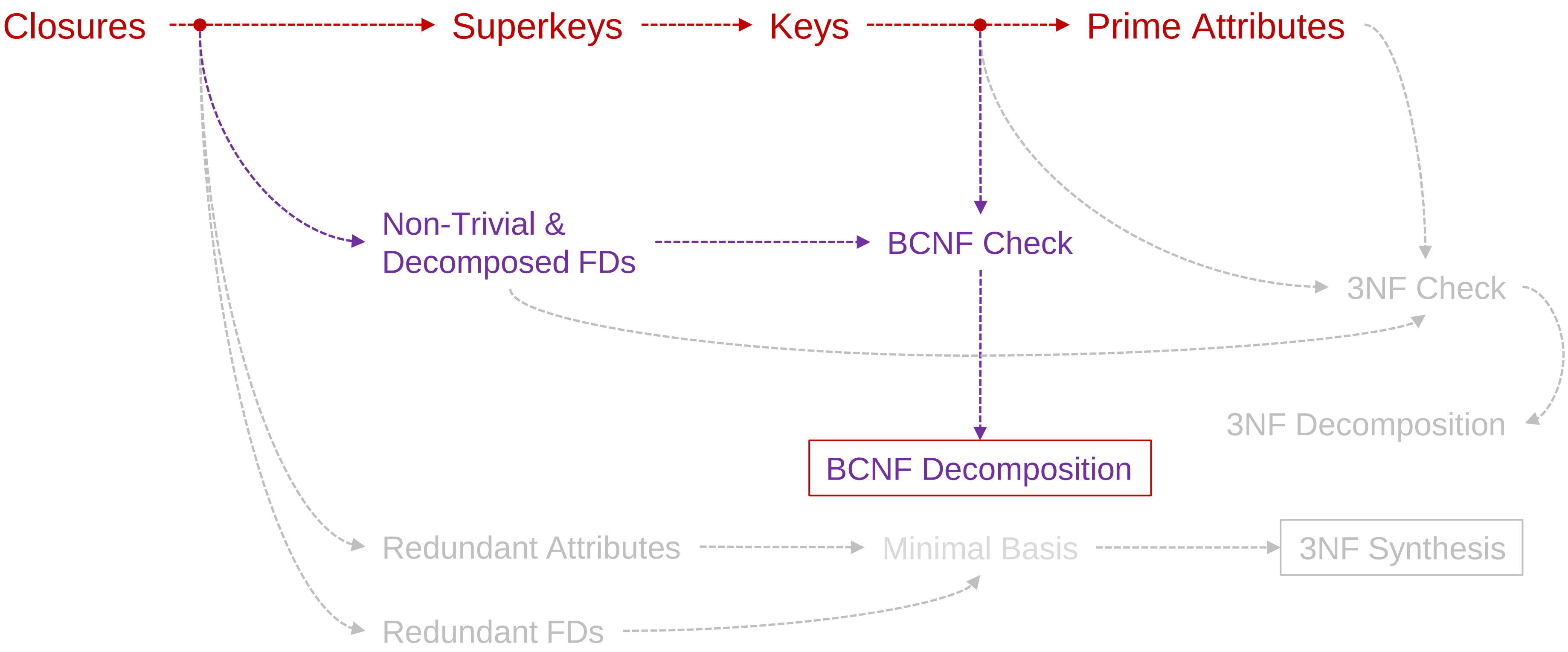
    ⇩
  - Check BCNF

    ⇩
  - Normalize to BCNF

❖ *Show that we can reconstruct the original table*



Algorithm Roadmap



Normal Form(s)

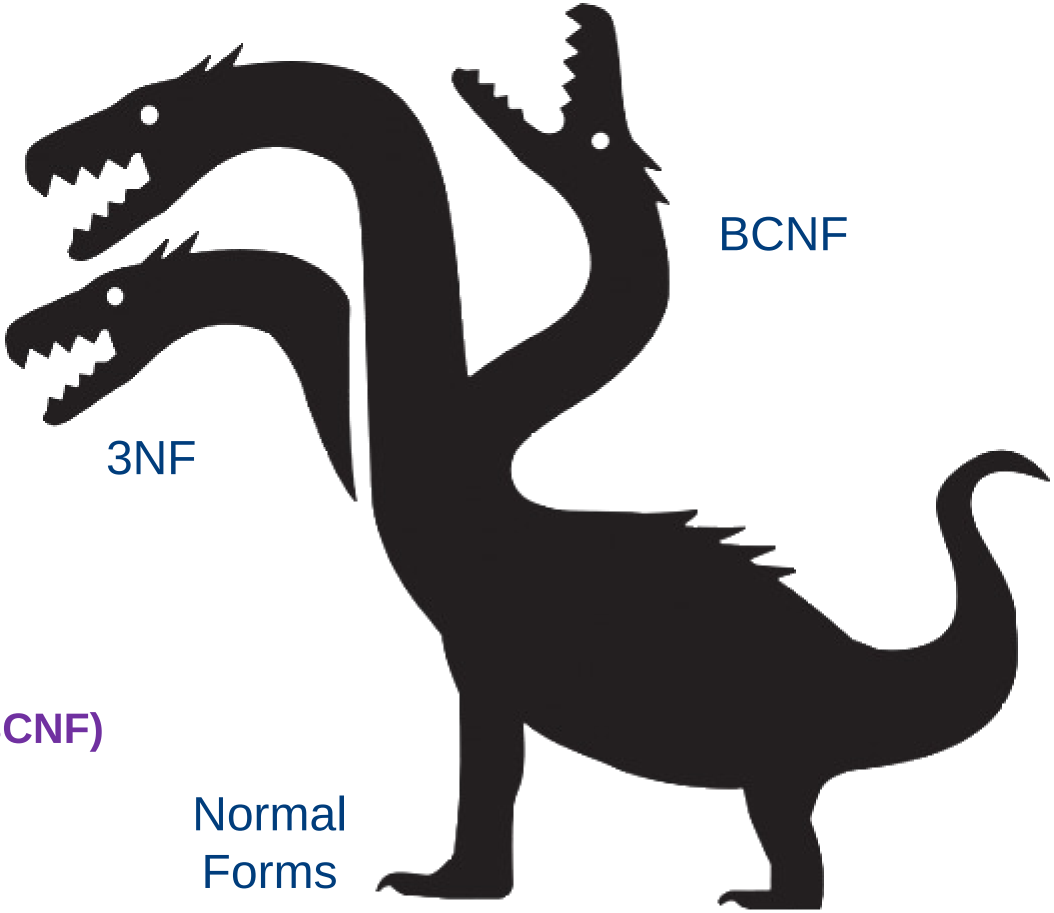
- What is it?
  - Conditions that a "good" table should satisfy
  - Various NFs in increasing order of strictness

Easy to satisfy. May have high redundancy.

1. 1^{st} NF
2. 2^{nd} NF
3. **3^{rd} NF (3NF)**
4. **Boyce-Codd NF (BCNF)**

Very little redundancy. Not always possible to satisfy.

5. 4^{th} NF
6. 5^{th} NF
7. 6^{th} NF



RECAP: FD

non-trivial and decomposed functional dependencies

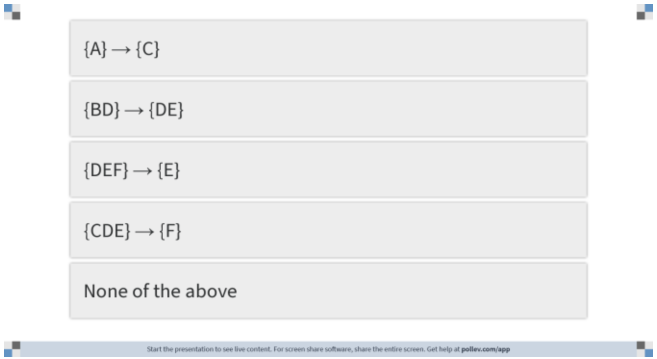
Recap FD ⇢ Define BCNF and Check ⇢ BCNF Decomposition ⇢ Table Reconstruction

Non-Trivial and Decomposed FD

- **Simplifying Discussion**
  - **Non-trivial** $\{A\} \rightarrow \{B\}$ where $\{B\} \not\subseteq \{A\}$ *(literally, not trivial)*
  - **Decomposed** $\{A\} \rightarrow \{B\}$ where B is a single attribute *(singular)*
  - A non-decomposed FD can always be transformed into the equivalent set of decomposed FD by *decomposition rule*
    - $\{BC\} \rightarrow \{DE\} \Leftrightarrow \{BC\} \rightarrow \{D\}$ and $\{BC\} \rightarrow \{E\}$

Non-Trivial and Decomposed FD

- **Simplifying Discussion**
  - **Non-trivial** $\{A\} \rightarrow \{B\}$ where $\{B\} \not\subseteq \{A\}$ *(literally, not trivial)*
  - **Decomposed** $\{A\} \rightarrow \{B\}$ where B is a single attribute *(singular)*
  - A non-decomposed FD can always be transformed into the equivalent set of decomposed FD by *decomposition rule*
    - $\{BC\} \rightarrow \{DE\} \Leftrightarrow \{BC\} \rightarrow \{D\}$ and $\{BC\} \rightarrow \{E\}$
  - **Exercise:** Which of these are non-trivial decomposed FDs?



Non-Trivial and Decomposed FD

- **Why?**
  - We will check normal forms based on the non-trivial and decomposed FDs on a table
    - Trivial FDs are not interesting *(i.e., trivial)* because they convey no information
    - Decomposed FDs are easier to reason and they are still valid
- **Deriving Non-Trivial and Decomposed FD from a Table**
  - Closure! *... non-trivialize, then decompose ...*
    1. Consider all subset of attributes in `R` *(do we need to consider `R`?)*
    2. Compute the closure of each subset
    3. Remove the “trivial” attributes
    4. Derive the decomposed FD from each closure

Non-Trivial and Decomposed FD

- **Deriving Non-Trivial and Decomposed FD from a Table**
  - **Example:** R(A, B, C) with {A} → {B}, {B} → {A} and {B} → {C}
    1. Consider all subset of attributes in R

       {A} {B} {C} {AB} {AC} {BC} {ABC}

    2. Compute the closure of each subset

       $\{A\}^+ = \{ABC\}$ $\{B\}^+ = \{ABC\}$ $\{C\}^+ = \{C\}$ $\{AB\}^+ = \{ABC\}$ $\{AC\}^+ = \{ABC\}$ $\{BC\}^+ = \{ABC\}$

    3. Remove the "trivial" attributes

       $\{A\}^+ = \{ABC\}$ $\{B\}^+ = \{ABC\}$ $\{C\}^+ = \{C\}$ $\{AB\}^+ = \{ABC\}$ $\{AC\}^+ = \{ABC\}$ $\{BC\}^+ = \{ABC\}$

    4. Derive the decomposed FD from each closure

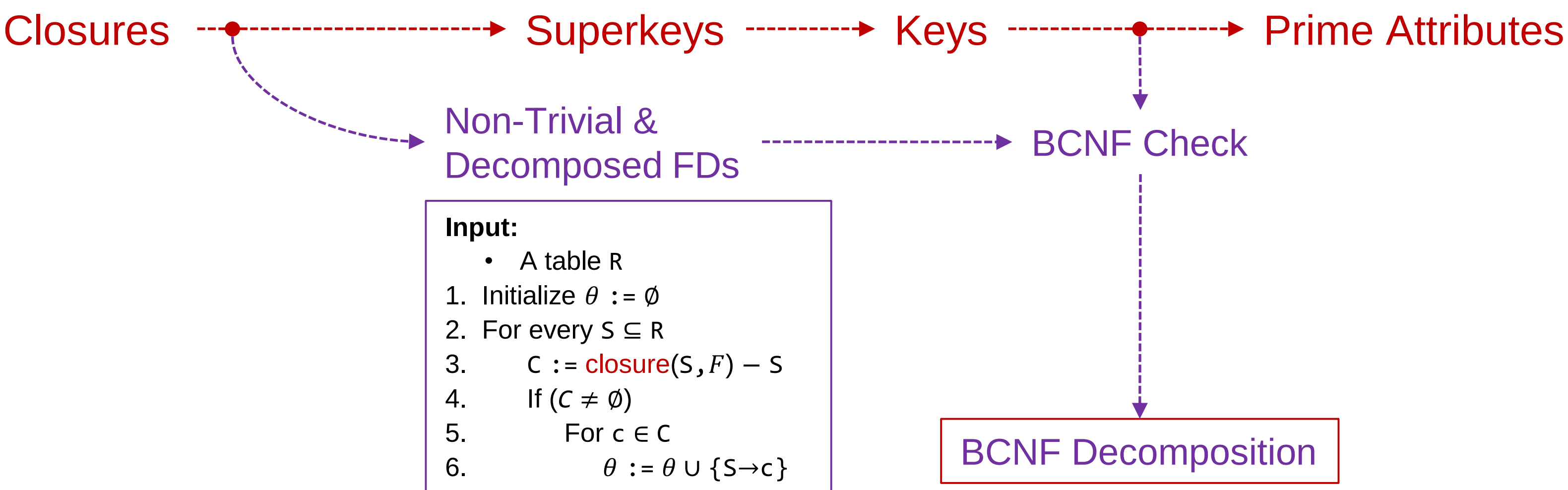
       {A} → {B} {A} → {C} {B} → {A} {B} → {C} {AB} → {C} {AC} → {B} {BC} → {A}

Non-Trivial and Decomposed FD

- **Deriving Non-Trivial and Decomposed FD from a Table**
  - **Exercise:** `R(A, B, C, D)` with $\{AB\} \rightarrow \{C\}$, $\{C\} \rightarrow \{D\}$ and $\{D\} \rightarrow \{A\}$

$\{A\}^+ =$	$\{B\}^+ =$	$\{C\}^+ =$	$\{D\}^+ =$
$\{AB\}^+ =$	$\{AC\}^+ =$	$\{AD\}^+ =$	
$\{BC\}^+ =$	$\{BD\}^+ =$	$\{CD\}^+ =$	
$\{ABC\}^+ =$	$\{ABD\}^+ =$	$\{ACD\}^+ =$	$\{BCD\}^+ =$

Algorithm Roadmap



Input:

- A table R

1. Initialize $\theta := \emptyset$
2. For every $S \subseteq R$
3. $\quad$ $C :=$ closure$(S, F) - S$
4. $\quad$ If $(C \neq \emptyset)$
5. $\quad\quad$ For $c \in C$
6. $\quad\quad\quad$ $\theta := \theta \cup \{S \rightarrow c\}$
7. Return θ

BCNF

Boyce-Codd Normal Form

Recap FD → Define BCNF and Check → BCNF Decomposition → Table Reconstruction

BCNF

- **Definition**
  - A table `R` is in BCNF if *every non-trivial and decomposed FD has a superkey as its left hand side*

  - **Example:** `R(A, B, C)` with `{A}` → `{B}`, `{B}` → `{A}` and `{B}` → `{C}`
    - Key: `{A}` and `{B}`
    - Non-trivial and decomposed FDs on `R`:
      - `{A}` → `{B}`
      - `{A}` → `{C}`
      - `{B}` → `{A}`
      - `{B}` → `{C}`
      - `{AB}` → `{C}`
      - `{AC}` → `{B}`
      - `{BC}` → `{A}`
    - For each of the above FD, the left hand side is a superkey

      ∴ `R` satisfies BCNF

BCNF

- **Definition**
  - A table `R` is in BCNF if *every non-trivial and decomposed FD has a superkey as its left hand side*

  - **Example:** `R(A, B, C)` with `{A} → {B}` and `{B} → {C}`
    - Key: `{A}`
    - Non-trivial and decomposed FDs on `R`:

      - `{A} → {B}` - `{A} → {C}` - `{B} → {C}`

      - `{AB} → {C}` - `{AC} → {B}`
    - B is **NOT** superkey in `{B} → {C}`

    ∴ `R` does **NOT** satisfy BCNF

BCNF

- **Intuition**
  - A table `R` is in BCNF if *every non-trivial and decomposed FD has a superkey as its left hand side*

  - **In other words,** any attribute `B` can depend ***only*** on superkeys
    - "In superkeys we trust"
    - Any dependency on non-superkeys is prohibited
  - **Why?**
    - Suppose B depends on non-superkey $C_1 C_2 \ldots C_n$
    - Since $C_1 C_2 \ldots C_n$ is not a superkey, the same $C_1 C_2 \ldots C_n$ may appear multiple times in the table *(why?)*
    - Whenever this happens, the same B would appear multiple times in the table

Table "Example"

C_1	C_2	$\cdots$	C_n	B

BCNF

- **Intuition**
  - A table `R` is in BCNF if *every non-trivial and decomposed FD has a superkey as its left hand side*
  - **Example:** The table shown on the right
    - Key: `{NRIC, Phone}`
    - FD that violates BCNF
      - `{NRIC}` → `{Name}`
      - `{NRIC}` → `{Address}`
    - Since `{NRIC}` is not superkey, the same `NRIC` can appear multiple times in the table
      - Every time `NRIC` is repeated, the corresponding `Name` and `Address` are also repeated

Table "Student_Data"

Name	NRIC	Phone	Address
Alice	1234	67899876	Jurong East
Alice	1234	83838484	Jurong East
Bob	5678	98765432	Pasir Ris

BCNF

- **Checking BCNF**
  - A table `R` is **NOT** in BCNF if *there exists at least one non-trivial and decomposed FD such that its left hand side is NOT superkey*

  - **By Counterexample:**

Can be done at the same time. Generate + check!

1. Consider all non-trivial and decomposed FDs of `R`
2. For each non-trivial and decomposed FDs of `R`, check that the left hand side is superkey
   - If not, then we have a counterexample
3. If no counterexample found, then `R` satisfies BCNF

BCNF

- **Checking BCNF**
  - A table R is **NOT** in BCNF if *there exists at least one non-trivial and decomposed FD such that its left hand side is NOT superkey*

  - **By Counterexample:**

Non superkey
1. Consider all subset of attributes of R
2. Compute the closure of each subset
3. Consider only the subset that are not superkey *(closure ≠ R)*

Non-trivial FD
4. Remove the “trivial” attributes
5. If the resulting set is non-empty, we have a counterexample

More but NOT All

Alternatively, $\{S\} \subset \{S\}^+ \subset R$

The FD violating BCNF are $\{S\} \rightarrow (\{S\}^+ - \{S\})$
and can be decomposed if needed

BCNF

- **Checking BCNF**
  - A table `R` is **NOT** in BCNF if *there exists at least one non-trivial and decomposed FD such that its left hand side is NOT superkey*
  - **Example:** `R(A, B, C, D)` with $\{AB\} \rightarrow \{C\}$, $\{C\} \rightarrow \{D\}$ and $\{D\} \rightarrow \{A\}$
    1. Consider all subset of attributes of `R`
    2. Compute the closure of each subset
    3. Consider only the subset that are not superkey *(closure ≠ R)*
    4. Remove the "trivial" attributes
    5. If the resulting set is non-empty, we have a counterexample

$\{A\}^+$	=	$\{A\}$	$\{B\}^+$	=	$\{B\}$	$\{C\}^+$	=	$\{ACD\}$	$\{D\}^+$	=	$\{AD\}$
$\{AB\}^+$	=	$\{ABCD\}$	$\{AC\}^+$	=	$\{ACD\}$	$\{AD\}^+$	=	$\{AD\}$			
$\{BC\}^+$	=	$\{ABCD\}$	$\{BD\}^+$	=	$\{ABCD\}$	$\{CD\}^+$	=	$\{ACD\}$			
$\{ABC\}^+$	=	$\{ABCD\}$	$\{ABD\}^+$	=	$\{ABCD\}$	$\{ACD\}^+$	=	$\{ACD\}$	$\{BCD\}^+$	=	$\{ABCD\}$

BCNF

- **Checking BCNF**
  - A table `R` is **NOT** in BCNF if *there exists at least one non-trivial and decomposed FD such that its left hand side is NOT superkey*

  - **Exercise:** `R(A, B, C, D)` with $\{B\} \to \{C\}$ and $\{B\} \to \{D\}$

$\{A\}^+ =$	$\{B\}^+ =$	$\{C\}^+ =$	$\{D\}^+ =$
$\{AB\}^+ =$	$\{AC\}^+ =$	$\{AD\}^+ =$	
$\{BC\}^+ =$	$\{BD\}^+ =$	$\{CD\}^+ =$	
$\{ABC\}^+ =$	$\{ABD\}^+ =$	$\{ACD\}^+ =$	$\{BCD\}^+ =$

BCNF

- **Checking BCNF**
  - A table R is **NOT** in BCNF if *there exists at least one non-trivial and decomposed FD such that its left hand side is NOT superkey*
  - **Exercise:** R(A, B, C, D) with {A} → {B}, {B} → {C}, {C} → {D} and {D} → {A}

$\{A\}^+ =$	$\{B\}^+ =$	$\{C\}^+ =$	$\{D\}^+ =$
$\{AB\}^+ =$	$\{AC\}^+ =$	$\{AD\}^+ =$	
$\{BC\}^+ =$	$\{BD\}^+ =$	$\{CD\}^+ =$	
$\{ABC\}^+ =$	$\{ABD\}^+ =$	$\{ACD\}^+ =$	$\{BCD\}^+ =$

BCNF

- **Checking BCNF**
  - A table R is **NOT** in BCNF if *there exists at least one non-trivial and decomposed FD such that its left hand side is NOT superkey*
  - **Exercise:** R(A, B, C, D) with {AB} → {D}, {BD} → {C}, {CD} → {A} and {AC} → {B}

$\{A\}^+$ =	$\{B\}^+$ =	$\{C\}^+$ =	$\{D\}^+$ =
$\{AB\}^+$ =	$\{AC\}^+$ =	$\{AD\}^+$ =	
$\{BC\}^+$ =	$\{BD\}^+$ =	$\{CD\}^+$ =	
$\{ABC\}^+$ =	$\{ABD\}^+$ =	$\{ACD\}^+$ =	$\{BCD\}^+$ =

BCNF

- **Checking BCNF**
  - A table R is **NOT** in BCNF if *there exists at least one non-trivial and decomposed FD such that its left hand side is NOT superkey*
  - **Exercise:** R(A, B, C, D, E) with {AB} → {C}, {C} → {E}, {E} → {A} and {E} → {D}

Algorithm Roadmap

Closures → Superkeys → Keys → Prime Attributes

Non-Trivial & Decomposed FDs → BCNF Check

BCNF Decomposition

Input:

- A table R

```
1. Initialize θ := ∅
2. For every S ⊆ R
3.     C := closure(S,F) − S
4.     If (C ≠ ∅)
5.         For c ∈ C
6.             θ := θ ∪ {S→c}
7. Return θ
```

Input:

- A table R
- A set of FD F on R

```
1. For every S ⊆ R
2.     C := closure(S,F)
3.     If (S ⊂ C ⊂ R)
4.         Return False
5. Return True
```

BCNF Decomposition

How to normalize a table to satisfy BCNF property

Recap FD → Define BCNF and Check → BCNF Decomposition → Table Reconstruction

BCNF Decomposition

- **Normalization**

Table "Student_Data"

Name	NRIC	Phone	Address
Alice	1234	67899876	Jurong East
Alice	1234	83838484	Jurong East
Bob	5678	98765432	Pasir Ris

Table "Student_Info"

Name	NRIC	Address
Alice	1234	Jurong East
Bob	5678	Pasir Ris

Table "Student_Contact"

NRIC	Phone
1234	67899876
1234	83838484
5678	98765432

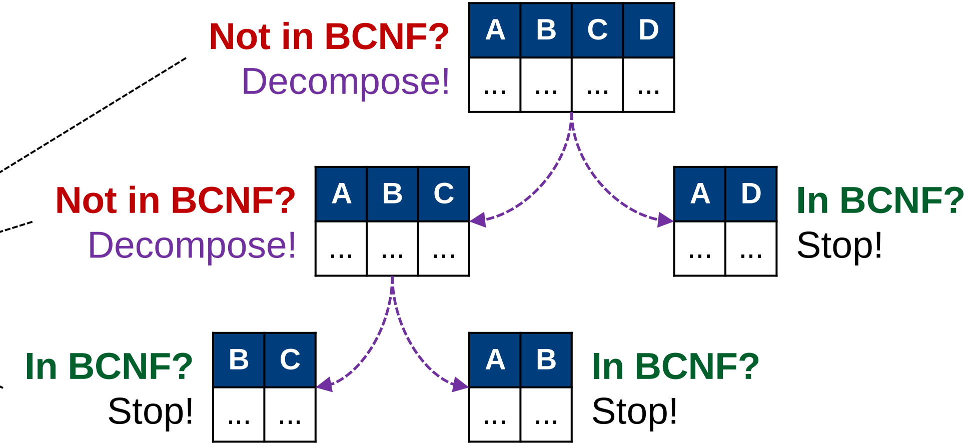
  - **Idea:**
    - Decompose until all are in BCNF
      - Split the attributes into another table and remove any duplicates
      - The splitting need to ensure that BCNF violations are removed

BCNF Decomposition

- **Normalization**
  - **Idea:**
    - Decompose until all are in BCNF
      - Split the attributes into another table and remove any duplicates
      - The splitting need to ensure that BCNF violations are removed

Input:
- A table `R`
- A set of FD F on `R`

1. For every `S` ⊆ `R`
2.   `C :=` closure(`S`)
3.   If ($S \subset C \subset R$)
4.   Return `False`
5. Return `True`



BCNF Decomposition

- **Normalization**
  - **Problem:**
    - How to decompose such that we remove at least one BCNF violation?
  - **Idea:**
    - Let's look at the violation again
      - Say `{B}` → `{CD}` violates BCNF property of `R(A, B, C, D)`
    - In what table does `{B}` → `{CD}` not violate BCNF property?
      - Make `B` a superkey!
      - `R1(B, C, D)`
    - ❖ We need to return the ***violation*** and not just `True/False`

Input:

- A table `R`
- A set of FD F on `R`

1. For every `S` ⊆ `R`
2. `C` := closure(`S`)
3. If ($S \subset C \subset R$)
4. Return **`False`**
5. Return **`True`**

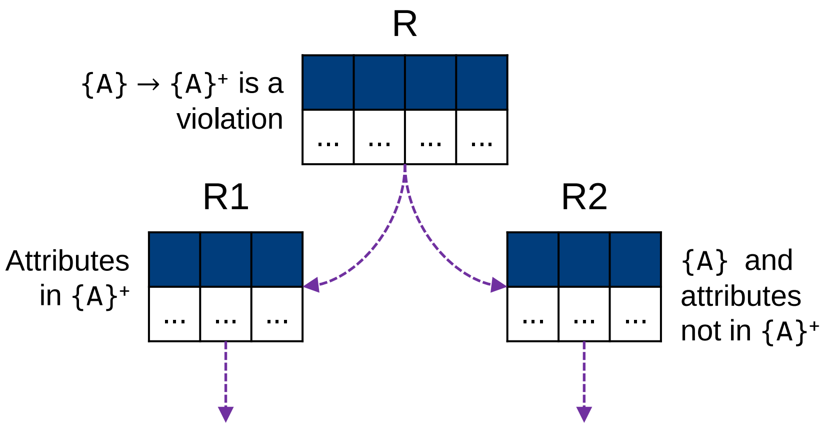
This is `True/False`, no information about violations

More but NOT All

Alternatively, `{S}` ⊂ `{S}`$^+$ ⊂ `R`

The FD violating BCNF are **`{S}` →(`{S}`$^+$ − `{S}`)** but **`{S}` → `{S}`$^+$** is sufficient

BCNF Decomposition



- **Normalization Algorithm**
  - **Input:**
    - A table `R`
    - A set of FD F on `R`

1. Check if `R` is in BCNF
   a. If not , there is an FD `{A}` → `{A}`$^+$ that violates it then `R` decompose into
      i. `R1`(*everything in* `{A}`$^+$)
      ii. `R2`(*everything in* `{A}` *as well as attributes in R but not in* `{A}`$^+$)
      iii. Recursively check `R1` and `R2`
      iv. Return the union of the result from recursive check
   b. Otherwise, stop
      i. Return `{R}`

Input:

- A table `R`
- A set of FD F on `R`

1. For every `S` ⊆ `R`
2. `C` := closure(`S`)
3. If ($S \subset C \subset R$)
4. Return **`{S}` → `C`**
5. Return **`NULL`**

BCNF Decomposition

- **Normalization Algorithm**
  - **Example:**
    - Known FD:
      - `{NRIC} → {Name, Address}`
  - **Steps:**
    1. Check BCNF property
       - `{NRIC} → {Name, NRIC, Address}` violates BCNF property

       a. Decompose `Student_Data` into two tables
          i. `Student_Info(Name, NRIC, Address)` *(`{NRIC}`$^+$)*
          ii. `Student_Contact(NRIC, Phone)` *(`{NRIC}` and attributes not in `{NRIC}`$^+$)*
          iii. Check if `Student_Info(Name, NRIC, Address)` and `Student_Contact(NRIC, Phone)` are in BCNF *(they are...)*

Table "Student_Data"

Name	NRIC	Phone	Address
Alice	1234	67899876	Jurong East
Alice	1234	83838484	Jurong East
Bob	5678	98765432	Pasir Ris

BCNF Decomposition

And that is how we got this decomposition

Table "Student_Data"

Name	NRIC	Phone	Address
Alice	1234	67899876	Jurong East
Alice	1234	83838484	Jurong East
Bob	5678	98765432	Pasir Ris

Table "Student_Info"

Name	NRIC	Address
Alice	1234	Jurong East
Bob	5678	Pasir Ris

Table "Student_Contact"

NRIC	Phone
1234	67899876
1234	83838484
5678	98765432

BCNF Decomposition

Collected Answers

1. `R1(A, B, C)` ✗
2. `R2(A, D)` ✓

- **Normalization Algorithm**
  - **Example:** `R(A, B, C, D)` with `{A}` → `{B}` and `{B}` → `{C}`
  - **Steps:**
    1. Check BCNF property
       - `{A} → {A, B, C}` violates BCNF property
       a. Decompose `R` into two tables
          i. `R1(A, B, C)` $(\{A\}^+)$
          ii. `R2(A, D)` $(\{A\} \cup (R - \{A\}^+))$
          iii. Check if `R1(A, B, C)` and `R2(A, D)` are in BCNF

NO! YES!

BCNF Decomposition

Collected Answers

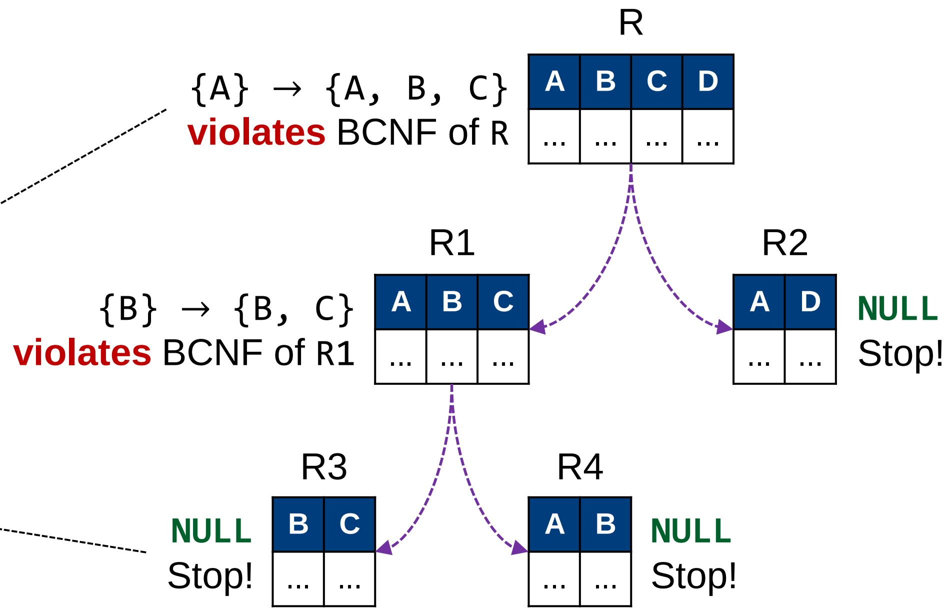
1. ~~R1(A, B, C)~~ ✗
   a. R3(B, C) ✓
   b. R4(A, B) ✓
2. R2(A, D) ✓

- **Normalization Algorithm**
  - **Example:** R(A, B, C, D) with {A} → {B} and {B} → {C}
  - **Steps:**
    1. Check BCNF property of R1(A, B, C) with {A} → {B} and {B} → {C}
       - {B} → {B, C} violates BCNF property
       a. Decompose R1 into two tables
          i. R3(B, C) *({B}⁺)*
          ii. R4(A, B) *({B} ∪ (R1 – {B}⁺))*
          iii. Check if R3(B, C) and R4(A, B) are in BCNF

YES! **YES!**

BCNF Decomposition

- **Normalization Algorithm**
  - **Example:** `R(A, B, C, D)` with `{A}` → `{B}` and `{B}` → `{C}`
    - Visualization



Input:
- A table `R`
- A set of FD F on `R`

1. For every `S` ⊆ `R`
2. `C :=` closure(`S`)
3. If ($S \subset C \subset R$)
4. Return **`{S} → C`**
5. Return **`NULL`**

BCNF Decomposition

- **Normalization Algorithm**
  - **Notes:**
    - The BCNF decomposition of a table may NOT be *unique*
      - Because any FD violating BCNF can be used for decomposition
    - If a table only has two attributes, then it is in BCNF
      - Why?
      - No need to check tables with only two attributes
  - **Issues:**
    - What happen if an FD has attributes *across* multiple tables?
    - Compute closure!
      - Then remove attributes that are irrelevant to the current table
    - Let's illustrate this with an example

BCNF Decomposition

Collected Answers

1. `R1(A, B)` ✓
2. `R2(A, C, D, E)` ?

- **Normalization Algorithm**
  - **Example:** `R(A,B,C,D,E)` with `{A}` → `{B}` and `{BC}` → `{D}`
  - **Steps:**
    1. Check BCNF property
       - `{A} → {A, B}` violates BCNF property

       a. Decompose `R` into two tables

          i. `R1(A, B)` *({A}⁺)*

          ii. `R2(A, C, D, E)` *({A} ∪ (R − {A}⁺))*

          iii. Check if `R1(A, B)` and `R2(A, C, D, E)` are in BCNF

YES! **???**

Do we even know what are the FDs that holds on R2?

`{A}` → `{B}`? But no `B`

`{BC}` → `{D}`? Also no `B`

BCNF Decomposition

Collected Answers

1. R1(A, B) ✓
2. R2(A, C, D, E) ×

- **Normalization Algorithm**
  - **Example:** R(A,B,C,D,E) with {A} → {B} and {BC} → {D}
  - **Steps:**
    1. Check BCNF property of R2(A, C, D, E) with {A} → {B} and {BC} → {D}
       - But all FD contains B and R2 has no attribute B
       a. Compute closure of each subset of attributes
       b. Remove attributes not in the current table *(projection)*
       - Now we can find the violation

$\{A\}^+$	=	{AB}	$\{C\}^+$	=	{C}	$\{D\}^+$	=	{D}	$\{E\}^+$	=	{E}
$\{AC\}^+$	=	{ABCD}	$\{AD\}^+$	=	{ABD}	$\{AE\}^+$	=	{AE}			
$\{CD\}^+$	=	{CD}	$\{CE\}^+$	=	{CE}	$\{DE\}^+$	=	{DE}			
$\{ACD\}^+$	=	{ABCD}	$\{ACE\}^+$	=	{ABCDE}	$\{ADE\}^+$	=	{ABDE}	$\{CDE\}^+$	=	{CDE}

BCNF Decomposition

Collected Answers

1. `R1(A, B)` ✓
2. ~~`R2(A, C, D, E)`~~ ✗
    a. `R3(A, D, E)` ✓
    b. `R4(A, C, E)` ✓

- ## Normalization Algorithm
  - **Example:** `R(A,B,C,D,E)` with `{A}` → `{B}` and `{BC}` → `{D}`
  - **Steps:**
    1. Check BCNF property of `R2(A, C, D, E)` with `{A}` → `{B}` and `{BC}` → `{D}`
       - `{A, C}` → `{A, C, D}` violates BCNF property

       a. Decompose `R2` into two tables
          i. `R3(A, C, D)` *({AC}⁺)*
          ii. `R4(A, C, E)` *({AC} ∪ (R2 – {AC}⁺))*
          iii. Check if `R3(A, C, D)` and `R4(A, C, E)` are in BCNF

YES! **YES!**

We used `{A}` → `{A, B}` as our FD violation.
But we can also use `{B, C}` → `{B, C, D}` as our FD violation!
Let's redo our normalization algorithm

BCNF Decomposition

Collected Answers

1. R1(B, C, D) ✓
2. R2(A, B, C, E) ✗

- **Normalization Algorithm**
  - **Example:** R(A,B,C,D,E) with {A} → {B} and {BC} → {D}
  - **Steps:**
    1. Check BCNF property
       - {B, C} → {B, C, D} violates BCNF property
       a. Decompose R into two tables
          i. R1(B, C, D) *({BC}⁺)*
          ii. R2(A, B, C, E) *({BC} ∪ (R − {BC}⁺))*
          iii. Check if R1(B, C, D) and R2(A, C, D, E) are in BCNF

YES! **NO!**

BCNF Decomposition

Collected Answers

1. R1(B, C, D) ✓
2. ~~R2(A, B, C, E)~~ ✗
   a. R3(A, B) ✓
   b. R4(A, C, E) ✓

- **Normalization Algorithm**
  - **Example:** R(A,B,C,D,E) with {A} → {B} and {BC} → {D}
  - **Steps:**
    1. Check BCNF property of R2(A, B, C, E) with {A} → {B} and {BC} → {D}
       - {A} → {A, B} violates BCNF property
       a. Decompose R2 into two tables
          i. R3(A, B) *({A}⁺)*
          ii. R4(A, C, E) *({A} ∪ (R2 – {A}⁺))*
          iii. Check if R3(A, B) and R4(A, C, E) are in BCNF

YES! **YES!**

Starting with {B, C} → {B, C, D}

1. R1(B, C, D)
2. R3(A, B)
3. R4(A, C, E)

Starting with {A} → {A, B}

1. R1(A, B)
2. R3(A, D, E)
3. R4(A, C, E)

Algorithm Roadmap

Closures → Superkeys → Keys → Prime Attributes

Non-Trivial & Decomposed FDs → BCNF Check → BCNF Decomposition

BCNF Check

Input:
- A table R
- A set of FD F on R

1. For every S ⊆ R
2.   C := project(R, F, S)
3.   If ($S \subset C \subset R$)
4.   Return **{S} → C**
5. Return **NULL**

Non-Trivial & Decomposed FDs

Input:
- A table R

1. Initialize θ := ∅
2. For every S ⊆ R
3.   C := closure(S, F) − S
4.   If ($C \neq \emptyset$)
5.   For c ∈ C
6. θ := θ ∪ {S→c}
7. Return θ

FD Projection

Input:
- A table R
- A set of FD F on R
- A set of attributes {A} ⊆ R

1. Initialize θ := ∅
2. For every S ⊆ R
3.   C := closure(S, F)
4. θ := θ ∪{{S} → (C∩A)}
5. Return θ

BCNF Decomposition

Input:
- A table R
- A set of FD F on R

1. f := BCNF Check(R, F)
2. If (f = NULL)
3.   Return {R}
4. {A}→{A}⁺ := f
5. Return BCNF(R1({A}⁺)) ∪ BCNF(R2({A}+(R−{A}⁺)))

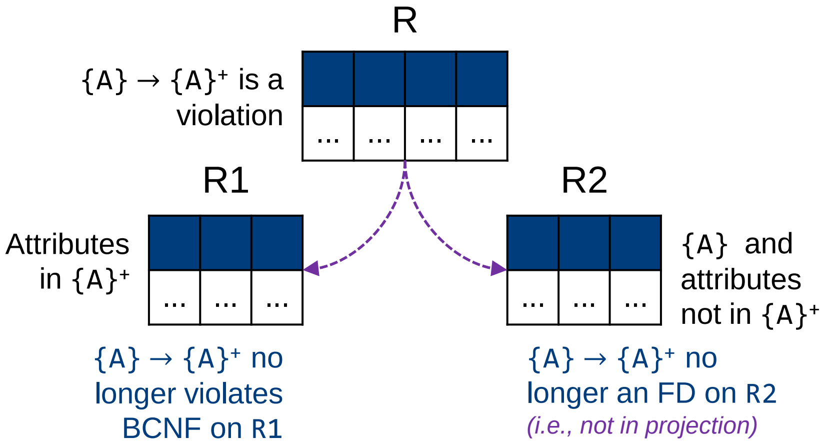
BCNF Decomposition

- **Normalization Algorithm**
  - Why does the BCNF decomposition algorithm work?
    - In other words, how can it eliminate violations of BCNF?

❖ In general, each decomposition step removes at least one BCNF violation

- Then we recursively removes all violation
- How do we know it will not add new violations?

➢ At the worst case, we have only two attributes ⇒ always BCNF



BCNF Decomposition

Simplified Algorithm

Find violation `{A}` → $\{A\}^+$

a. `R1(`$\{A\}^+$`)`

b. `R2({A}` ∪ `(R-{A}+))`

Repeat with `R1` and `R2`

- **Normalization Algorithm**
  - **Exercise:** `R(A, B, C, D)` with `{A}` → `{B}` and `{A}` → `{C}`
  - **Steps:**

BCNF Decomposition

Simplified Algorithm

Find violation `{A} → {A}+`

a. `R1({A}+)`

b. `R2({A} ∪ (R-{A}+))`

Repeat with `R1` and `R2`

- **Normalization Algorithm**
  - **Exercise:** `R(A, B, C, D)` with `{BC}` → `{D}`, `{D}` → `{A}` and `{A}` → `{B}`
  - **Steps:**

BCNF Decomposition

- **Properties of BCNF**
  - **Good** properties
    - No update or deletion or insertion anomalies
    - Small redundancies *(very hard to completely remove redundancies)*
    - The original table can always be reconstructed from the decomposed tables
      - How?
      - `SELECT * FROM Student_Info, Student_Contact`
        `WHERE Student_Info.NRIC = Student_Contact.NRIC`

This is called **"Lossless Join"**

```
SELECT * FROM Student_Info, Student_Contact
WHERE  Student_Info.NRIC = Student_Contact.NRIC
```

Table "Student_Data"

Name	NRIC	Phone	Address
Alice	1234	67899876	Jurong East
Alice	1234	83838484	Jurong East
Bob	5678	98765432	Pasir Ris

Table "Student_Info"

Name	NRIC	Address
Alice	1234	Jurong East
Bob	5678	Pasir Ris

Table "Student_Contact"

NRIC	Phone
1234	67899876
1234	83838484
5678	98765432

BCNF Decomposition

- **Lossless Join Decomposition**
  - Say we decompose a table `R` into two tables `R1` and `R2`
  - The decomposition guarantees ***lossless join***, whenever the common attributes in `R1` and `R2` constitute a superkey of `R1` or `R2`
    - **Example:**
      - `R(A, B, C)` decomposed into `R1(A, B)` and `R2(B, C)` with `B` being a superkey of `R2`
      - `R(A, B, C, D)` decomposed into `R1(A, B, C)` and `R2(B, C, D)` with `BC` being a superkey of `R1`

BCNF Decomposition

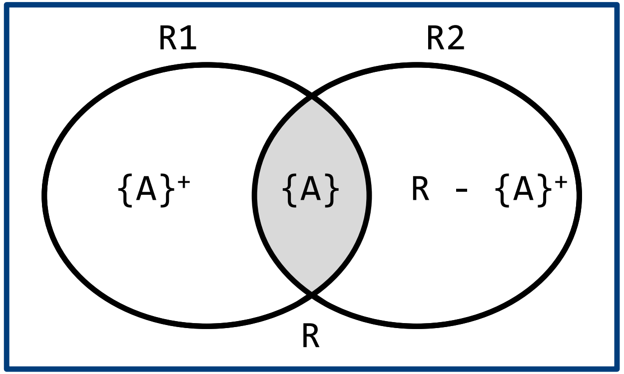
Lossless Join

The decomposition guarantees lossless join, whenever the *common attributes* in R1 and R2 constitute a superkey of R1 or R2

- **Lossless Join Decomposition of BCNF**
  - BCNF decomposition ***guarantee*** lossless join decomposition
  - **Why?**
    - Consider a violation $\{A\} \rightarrow \{A\}^+$
    - Decomposed into $R1(\{A\}+)$ and $R2(\{A\} \cup (R - \{A\}^+))$
    - **Common attributes?**
      - $\{A\}^+ \cap (\{A\} + (R - \{A\}^+))$

        $\Rightarrow \{A\}$
    - But $\{A\} \rightarrow \{A\}^+$ and $R1(\{A\}+)$
      - So $\{A\}$ is a superkey of R1

      $\therefore$ This decomposition guarantees lossless join decomposition



BCNF Decomposition

- **Properties of BCNF**
  - **Good** properties
    - No update or deletion or insertion anomalies
    - Small redundancies *(very hard to completely remove redundancies)*
    - The original table can always be reconstructed from the decomposed tables
  - **Bad** properties
    - Dependencies may not be preserved *(non dependency-preserving decomposition)*
      - *Topic for next week lecture*

QUESTION?