Relational Database Design

- Features of Good Relational Design
- Atomic Domains and First Normal Form
- Decomposition Using Functional Dependencies
- Functional Dependency Theory
- Algorithms for Functional Dependencies
- More Normal Form
- Database-Design Process

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

DEPARTMENT

Dname	Dnumber	Dmgr_ssn
Research	5	333445555
Administration	4	987654321
Headquarters	1	888665555

Redundancy

EMP_DEPT

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

Features of a Good relational database design

- Minimum Redundancy
  - Storage of same data in more than one location - **redundancy**
  - Disadvantage: Wastage of storage space
  - Results in updation anomalies
- Fewer null values in tuples
  - Values of all the attributes of a tuple are not known, so null value is to be stored
  - Cannot be provided to primary key

Update Anomalies

Storing natural joins of base relations leads to an additional problem referred to as **update anomalies**. These can be classified into insertion anomalies, deletion anomalies, and modification anomalies.

- Insertion anomaly: It leads to a situation in which certain information cannot be inserted into a relation unless some other information is stored
- Deletion anomaly: It leads to a situation in which deletion of data representing certain information results in losing data representing some other information that is associated with it
- Modification Anomaly: It leads to a situation in which repeated data changed at one place results in inconsistency unless the same data are also changed at other places

Update Anomalies-Example

- Insertion Anomaly
- To insert a new employee tuple into EMP_DEPT, we must include either the attribute values for the department that the employee works for, or NULL.
- For example, to insert a new tuple for an employee who works in department number 5, we must enter all the attribute values of department 5 correctly so that they are *consistent* with the corresponding values for department 5 in other tuples in EMP_DEPT.
- It is difficult to insert a new department that has no employees as yet in the EMP_DEPT relation. The only way to do this is to place NULL values in the attributes for employee.

Update Anomalies-Example

- Deletion Anomaly
- If we delete from EMP_DEPT an employee tuple that happens to represent the last employee working for a particular department, the information concerning that department is lost from the database
- Modification Anomaly
- In EMP_DEPT, if we change the value of one of the attributes of a particular department—say, the manager of department 5—we must update the tuples of *all* employees who work in that department; otherwise, the database will become inconsistent
- If we fail to update some tuples, the same department will be shown to have two different values for manager in different employee tuples, which would be wrong

Update Anomalies-Example

Three anomalies are undesirable and cause difficulties to

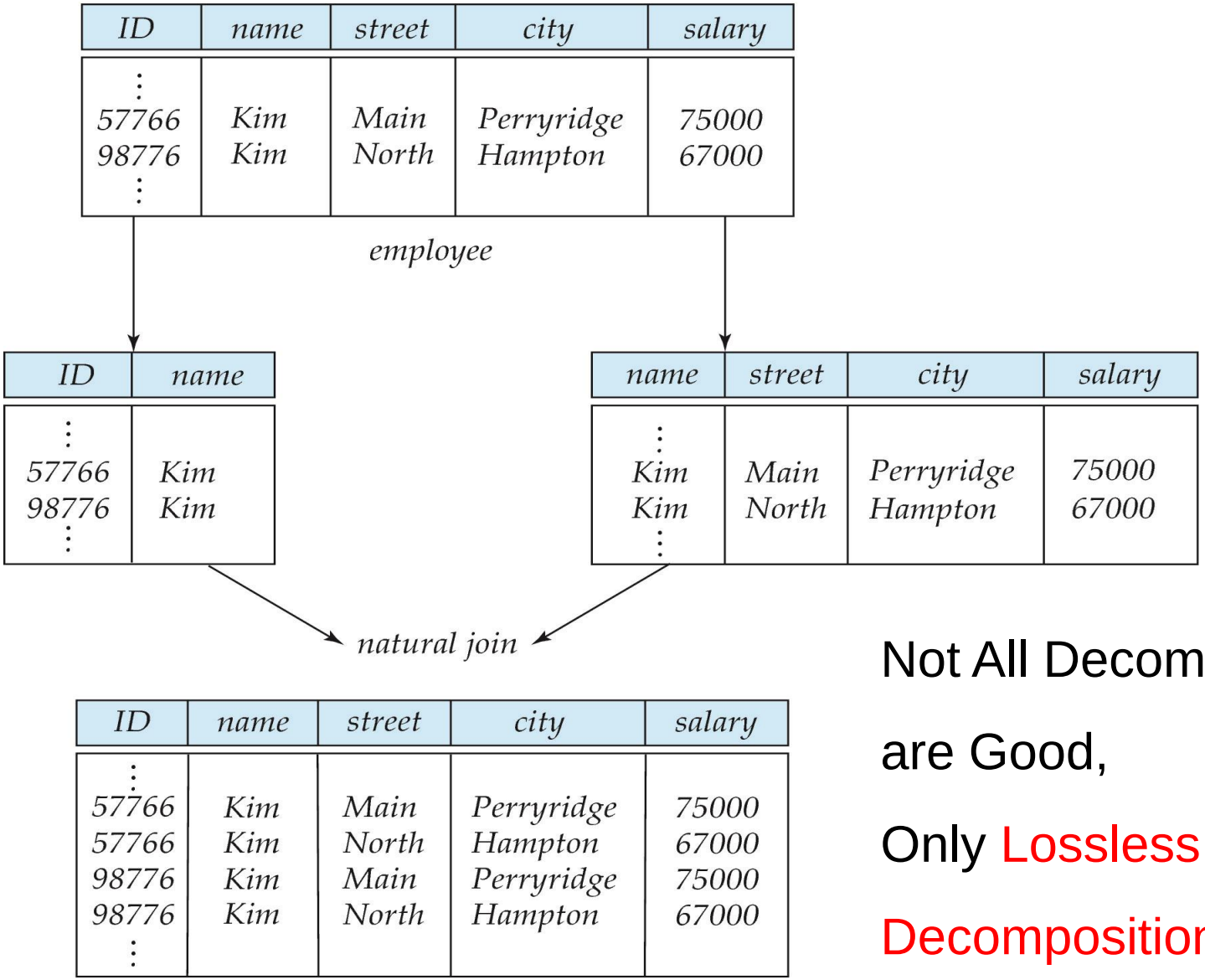
- maintain consistency of data as well as require unnecessary updates that can be avoided

Decomposition of a Relation

- While designing a relational database certain problems that are met due to redundancy and null values can be resolved by decomposing a relation.
- In decomposing, a relation is replaced with a collection of smaller relations with specific relationship between them
- Formally, the **decomposition** of a relation schema **R** is defined as its replacement by a set of relation schemas- $R_1, R_2, \ldots, R_n$ such that each relation schema contains a subset of attributes of **R**.

How to avoid redundancy & achieve Lossless Decomposition ?

A Lossy Decomposition



Not All Decompositions are Good,

Only Lossless Decomposition is Good

Example of Lossless-Join Decomposition

- **Lossless join decomposition**
- Decomposition of $R = (A, B, C)$

  $R_1 = (A, B)$ $R_2 = (B, C)$

A	B	C
α	1	A
β	2	B

r

A	B
α	1
β	2

$\Pi_{A,B}(r)$

B	C
1	A
2	B

$\Pi_{B,C}(r)$

$\Pi_A (r) \bowtie \Pi_B (r)$

A	B	C
α	1	A
β	2	B

Goal — Devise a Theory for the Following

- Decide whether a particular relation *R* is in "good" form.
- In the case that 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-join decomposition**
- Our theory is based on:
  - **functional dependencies**

Functional Dependency

- ***Functional dependency is a relationship between two attributes***.
- Functional dependencies are the special forms ***of integrity constraints that generalize the concepts of keys***
- They play a key role in developing a good database schema

Functional Dependency

- A functional dependency is a constraint between two sets of attributes from the database
- A **functional dependency**, denoted by $X \rightarrow Y$, between two sets of attributes X and Y that are subsets of R specifies a *constraint* on the possible tuples that can form a relation state r of R.
- The constraint is that, for any two tuples $t1$ and $t2$ in r that have $t1[X] = t2[X]$, they must also have $t1[Y] = t2[Y]$.
- This means that the values of the Y component of a tuple in r depend on, or are *determined by,* the values of the X component; alternatively, the values of the X component of a tuple uniquely (or **functionally**) *determine* the values of the Y component.

Functional Dependencies (Cont.)

- We also say that there is a functional dependency from *X* to *Y*, or that *Y* is **functionally dependent** on *X*. The abbreviation for functional dependency is **FD** or **f.d.**
- The set of attributes *X* is called the **left-hand side (determinant)** of the FD, and *Y* is called the **right-hand side (dependent)**
- Thus, *X* functionally determines *Y* in a relation schema *R* if, and only if, whenever two tuples of *r*(*R*) agree on their *X*-value, they must necessarily agree on their *Y*-value.

Functional Dependencies (Cont.)

- If a constraint on *R* states that there cannot be more than one tuple with a given *X*-value in any relation instance *r*(*R*)—that is, *X* is a **candidate key** of *R*—this implies that $X \rightarrow Y$ for any subset of attributes *Y* of *R* (because the key constraint implies that no two tuples in any legal state *r*(*R*) will have the same value of *X*).
- If *X* is a candidate key of *R*, then $X \rightarrow R$.
- If $X \rightarrow Y$ in *R*, this does not say whether or not $Y \rightarrow X$ in *R*.

Functional Dependencies (Cont.)

Table Student

studID	lastname	status	credits
1234567	Smith	Undergraduate	254
1234568	Don	Undergraduate	149
1234569	Dallas	Graduate	543

Functional dependencies include:

{stuId}→{lastName}

{stuId} →{lastName, credits, status, stuId}

{credits}→{status} (but not {status}→{credits})

Functional Dependencies (Cont.)

R(A,B,C,D)

A	B	C	D
a1	b1	c1	d1
a1	b2	c2	d2
a2	b2	c2	d3
a3	b3	c4	d3

The following FDs *may hold* :

$B \rightarrow C$; $C \rightarrow B$; $\{A, B\} \rightarrow C$; $\{A, B\} \rightarrow D$; and $\{C, D\} \rightarrow B$.

The FDs that *Do not* hold because we already have violations of them in the given extension: $A \rightarrow B$ (tuples 1 and 2 violate this constraint);

$C \rightarrow D$, $B \rightarrow A$ (tuples 2 and 3 violate this constraint);

$D \rightarrow C$ (tuples 3 and 4 violate it).

Functional Dependencies (Cont.)

Relation r

A	B	C	D
a_1	b_1	c_1	d_1
a_1	b_2	c_1	d_2
a_2	b_2	c_2	d_2
a_2	b_3	c_2	d_3
a_3	b_3	c_2	d_4

List the possible functional dependencies.

A →C
{A,B}→C
D → B

Functional Dependencies

Example

Assume **ITEMS(it_name, comp_name, price**) is a schema storing items(toothbrush, toothpaste etc.) and company(Colgate, Pepsodent etc.) names and price.

It_name	Comp_name	price
Toothpaste	**Colgate**	20
Toothpaste	**Pepsodent**	30
Toothbrush	**Colgate**	55
Toothbrush	**Pepsodent**	90

Functional Dependencies

Example

Assume **ITEMS(it_name, comp_name, price**) is a schema storing items(toothbrush, toothpaste etc.) and company(Colgate, Pepsodent etc.) names and price.

It_name	Comp_name	price
Toothpaste	**Colgate**	20
Toothpaste	**Pepsodent**	30
Toothbrush	**Colgate**	55
Toothbrush	**Pepsodent**	90

An It_name may be produced by many companies ,

therefore **It_name $\not\rightarrow$ Comp_Name**

($\not\rightarrow$ **means NOT Functionally depends**)

A company may be producing many It_name,

therefore **Comp_Name $\not\rightarrow$ It_name**

Functional Dependencies

Example (Contd.)

It_name	Comp_name	price
Toothpaste	Colgate	20
Toothpaste	Pepsodent	30
Toothbrush	Colgate	55
Toothbrush	Pepsodent	90

Similarly price is not functionally depends on It_name or Comp_name

It_name $\not\rightarrow$ Price & Comp_Name $\not\rightarrow$ Price

However a Price is uniquely determined by the combination of (It_Name,Comp_Name), therefore

(It_Name,Comp_Name) $\rightarrow$ Price

Trivial Functional Dependency

- *A* functional dependency is **trivial** if it is satisfied by all relations
  - **X → X** is satisfied by all relations having attribute X.
  - Similarly, **XY →X** is satisfied by all relations having attributes X and Y.
- Example*:*
  - *ID, name → ID*
  - *name → name*

In general, $\alpha \rightarrow \beta$ is trivial if $\beta \subseteq \alpha$

- The dependencies that are not trivial are called **non-trivial dependencies.**
- These are the dependencies that correspond to the essential integrity constraints.

Functional Dependencies
Super Key & Candidate key(Minimal Super key)

- ***K*** is a super key for relation schema ***R*** if and only if $K \rightarrow R$
- ***K*** is a candidate key (minimal Super Key) for ***R*** if and only if following **2** conditions hold.
  - $K \rightarrow R$ *(means **K** is **Super key**)*, and
  - There exists **no** subset of **K** **(**i.e. for **none** of $\alpha \subset K$**)** $\alpha \rightarrow R$ ***holds***
    - *Means **for all subsets of K,** $\alpha \rightarrow R$ should **not hold**.*

Example: Minimal Super Key

- *Consider the* **ITEMS** *relation.*

  Let us take K= **(It_Name,Comp_Name)**

A Price is uniquely determined by the combination of (It_Name,Comp_Name), therefore

(It_Name,Comp_Name) → Price

Following two functional dependency also holds because of trivial functional dependency (i.e. $\alpha \rightarrow \beta$ holds always if $\beta \subset \alpha$)

(It_Name,Comp_Name) → It_Name **(because trivial FD)**

because It_Name ⊂ (It_Name,Comp_Name)

(It_Name,Comp_Name) → Comp_Name **(because trivial FD)**

because Comp_Name⊂ (It_Name,Comp_Name)

Therefore (It_Name,Comp_Name) → ITEMS

(It_Name,Comp_Name) is Super Key for ITEMS

Is it Minimal Super key also?

Example: Minimal Super Key (Contd.)

As Discussed in previous slide **(It_name, Comp_name) is Super Key.**

- Further **K** i.e. **(It_name, Comp_name)** can be a **candidate key** (means minimal Super key) ,

  if **none of the subset of K *determine* R** i.e. **ITEMS.**

"None of the sub sets of (It_name, Comp_name) is Super key"

Subsets of (It_name, Comp_name) are

(It_name) & (Comp_Name)

We know **It_name alone do not determine ITEMS & Comp_Name alone also do not determine ITEMS .**

i.e. Neither It_name nor Comp_Name is Super Key for ITEMS

- **i.e** **It_name** $\not\rightarrow$ **ITEMS** & **Comp_name** $\not\rightarrow$ **ITEMS**

Example: Minimal Super Key (Contd.)

- From Super key, **(It_name, Comp_name)** we are unable to find **any subset** which **determine ITEMS** .

not possible to find any sub set of

(**It_name, Comp_name) which is super key**

- i.e. Hence **(It_name, Comp_name)** are **minimum attributes required** to determine ITEMS.

- Therefore **(It_name, Comp_name) is minimal Super key (i.e. candidate key)**

Example: minimal Super Key

R={A,B,C,D}

A	B	C	D
a_1	b_1	c_1	d_1
a_1	b_2	c_1	d_2
a_2	b_2	c_2	d_2
a_2	b_3	c_2	d_3
a_3	b_3	c_2	d_4

Is (A,B) → D Super key?

If yes **, is it minimal Super Key also?**

Yes, **(A,B) → D** is **Super Key.**

How to check (A,B) is also minimal Super key?

- None subsets of (A,B) should determine D

Subsets are – A, B

Check A→ D ? B→ D ?

If none of these Dependency hold then **AB** **is Minimal S.Key**

If any one of these Functional Dependency holds then **AB** is **not Minimal.S.Key**

A→ D NO, **B→ D** NO. No **subsets of {A,B} determines D**

Means **(A,B) → D** **Super key**, also **minimal Super key.**

Note: that Functional dependency or minimal super key is not determined by the only data appearing in a relation at a given point of time, instead it **depends on the meaning (semantics) of the attributes.**

Functional Dependencies(Cont'd)

Full Functional dependency:

- If A and B are attributes of a table, B is **fully functionally dependent** on A if B is functionally dependent on A, but not on **any proper subset of A**.

- **Example**
  Consider the example **ITEMS(It_Name, Comp_Name, Price)**

- Consider Functional Dependency **(It_Name, Comp_Name) → Price**
- **Price** is **Fully Functionally dependent** because-

- It_Name $\not\rightarrow$ Price
- Comp_Name) $\not\rightarrow$ Price

} Price do not Functionally dependent on none of the proper subset of (It_Name, Comp_Name)

Functional Dependencies(Cont'd)

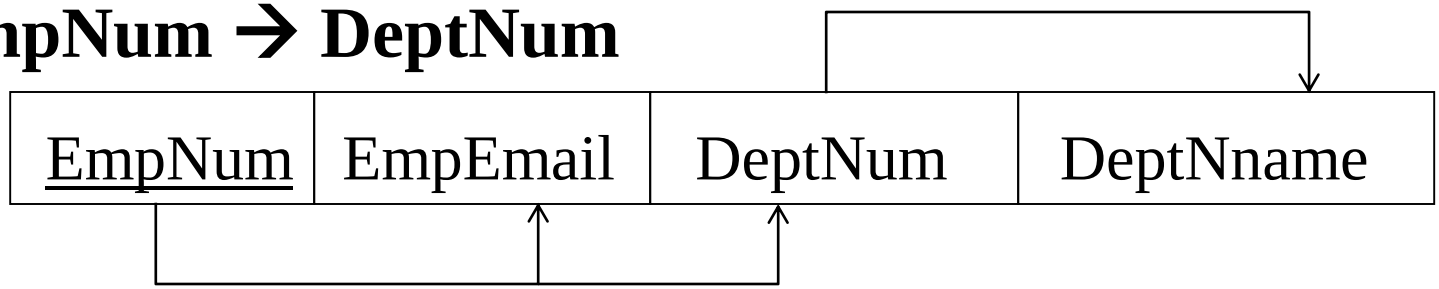
Partial Functional Dependency:

- If A and B are attributes of a table, B is **partially dependent** on A if there is some attribute that can be removed from A and yet the dependency still holds.
- **Example**

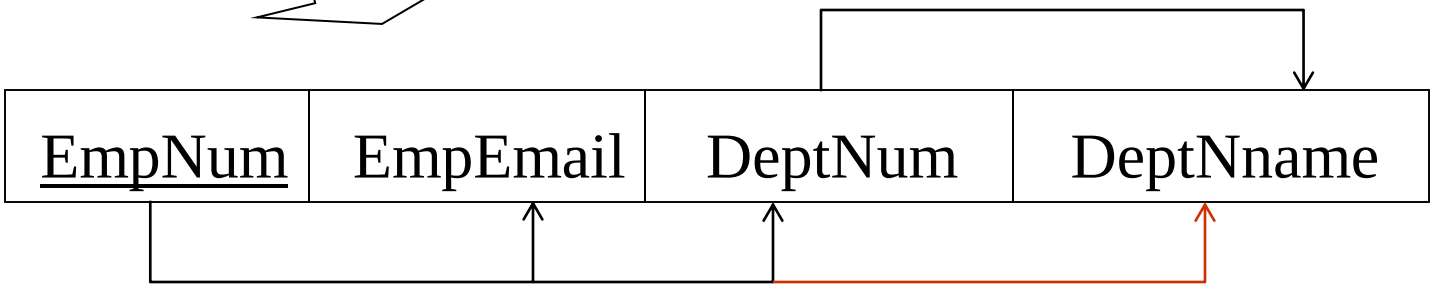
  (Empno,Name) → Deptno , but ***Empno → Deptno***
- Deptno is also dependent on subset Empno of (Empno,Name),
- Therefore **Deptno** is **not fully dependent** on (Empno,Name),
- Means **Deptno** is **Partially Dependent** on **(Empno, Name)**

Transitive dependency

EmpNum → DeptNum



DeptNum → DeptName



DeptName is *transitively dependent* on EmpNum via DeptNum

EmpNum → DeptName

Closure of a Set of Functional Dependencies

- Given a set ***F*** of functional dependencies, there are certain **other functional dependencies** that are **logically implied by *F*.**
  - For example: If $A \rightarrow B$ and $B \rightarrow C$, then we can infer that $A \rightarrow C$ (**transitivity*), $A \rightarrow AC$(**augmentation*), $AB \rightarrow B$ (trivial). $AB \rightarrow BC$ (**augmentation*)
- The set of **all** functional dependencies logically implied by *F* is the **closure** of *F* .
- We denote the ***closure* of *F*** by $\mathbf{F^+}$.
- $\mathbf{F^+}$ is a superset of ***F***. $\mathbf{F^+} \supset \boldsymbol{F}$

**** are discussed in next slides(Armstrong's Axioms)***

Closure of a Set of Functional Dependencies

- We can find F^+, the closure of F, by repeatedly applying **Armstrong's Axioms:**
  - if $\beta \subseteq \alpha$, then $\alpha \rightarrow \beta$ **(reflexivity)**
  - if $\alpha \rightarrow \beta$, then $\gamma\ \alpha \rightarrow \gamma\ \beta$ **(augmentation)**
  - if $\alpha \rightarrow \beta$, and $\beta \rightarrow \gamma$, then $\alpha \rightarrow \gamma$ **(transitivity)**
- These rules are
  - **sound** (generate only functional dependencies that actually valid), and
  - **complete** (generate all functional dependencies that hold).

Closure of Functional Dependencies (Cont.)

- **Additional rules:**
  1. If $\alpha \rightarrow \beta$ holds *and* $\alpha \rightarrow \gamma$ holds, then $\alpha \rightarrow \beta\gamma$ holds **(union)**
  2. If $\alpha \rightarrow \beta\ \gamma$ holds, then $\alpha \rightarrow \beta$ holds and $\alpha \rightarrow \gamma$ holds **(decomposition)**
  3. If $\alpha \rightarrow \beta$ holds *and* $\gamma\beta \rightarrow \delta$ holds, then $\alpha\gamma \rightarrow \delta$ holds **(pseudotransitivity)**

  The above rules can be inferred from Armstrong's axioms.

* For Proof, see notes section.

Closure of Functional Dependencies (Cont.)

- **Additional rules:**
  1. If $\alpha \rightarrow \beta$ holds *a*nd $\alpha \rightarrow \gamma$ holds, then $\alpha \rightarrow \beta\gamma$ holds **(union)**

Union rule.- Proof

$\alpha \rightarrow \beta$	given
$\alpha\alpha \rightarrow \alpha\beta$	*augmentation rule (augmenting α on both sides)*
$\alpha \rightarrow \alpha\beta$	*union of identical sets* $\alpha\alpha = \alpha$ ---(1)
$\alpha \rightarrow \gamma$	given
$\alpha\beta \rightarrow \beta\gamma$	*augmentation rule(augment β on both side* ---(2)
$\alpha \rightarrow \beta\gamma$	*transitivity rule (from (1) & (2))* Proved

Closure of Functional Dependencies (Cont.)

- **Additional rules:**
  2. If $\alpha \rightarrow \beta\ \gamma$ holds, then $\alpha \rightarrow \beta$ holds and $\alpha \rightarrow \gamma$ holds **(decomposition)**

Decomposition rule.- Proof

$\alpha \rightarrow \beta\gamma$	given (1)
$\beta\ \gamma \rightarrow \beta$	*(2) reflexivity rule, because* $\beta \subseteq \beta\ \gamma$
$\alpha \rightarrow \beta$	*(1) & (2) transitivity rule (proved)*
$\beta\ \gamma \rightarrow \gamma$	*(3) reflexive rule, because* $\gamma \subseteq \beta\ \gamma$
$\alpha \rightarrow \gamma$	*(1) & (3) transitive rule (proved)*

Closure of Functional Dependencies (Cont.)

- **Additional rules:**
  3. If $\alpha \rightarrow \beta$ holds and $\gamma\beta \rightarrow \delta$ holds, then $\alpha\gamma \rightarrow \delta$ holds (**pseudo transitivity**)

Pseudo transitivity -Proof

$\alpha \rightarrow \beta$	*given*
$\alpha\gamma \rightarrow \gamma\beta$	*augmentation rule augmenting* γ *on both side*
$\gamma\beta \rightarrow \delta$	*given*
$\alpha\gamma \rightarrow \delta$	*transitivity rule(Proved)*

Example: **Finding some logically implied functional dependencies using Armstrong Axioms**

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

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

if $\beta \subseteq \alpha$, **then** $\alpha \rightarrow \beta$
(reflexivity)
if $\alpha \rightarrow \beta$, **then** $\gamma\ \alpha \rightarrow \gamma\ \beta$
(augmentation)
if $\alpha \rightarrow \beta$, **and** $\beta \rightarrow \gamma$, **then** $\alpha \rightarrow \gamma$
(transitivity)

- some members of F^+
  - $A \rightarrow H$
    - by transitivity from $A \rightarrow B$ *and* $B \rightarrow H$
  - $AG \rightarrow I$
    - Since $A \rightarrow C$ and $CG \rightarrow I$ *(pseudotransitivity)*
  - $CG \rightarrow HI$
    - *Since* $CG \rightarrow H$ *and* $CG \rightarrow$ I (Union rule)

Prove: AB → CH & A → BC

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 f_1 and f_2 in F^+

    if f_1 and f_2 can be combined using transitivity

      then add the resulting functional dependency to F^+

until F^+ does not change any further

Closure of Attribute Sets

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

Algorithm to **compute a^+**, the **closure** of **a** 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
```

Example for Attribute Set Closure

- $R = (A, B, C, G, H, I)$
- $F = \{ A \rightarrow B$
  $A \rightarrow C$
  $CG \rightarrow H$
  $CG \rightarrow I$
  $B \rightarrow H \ \}$ Find AG^+
- $(AG)^+$
  1. $result = AG$
  2. $result = ABCG$ $(A \rightarrow C$ and $A \rightarrow B)$
  3. $result = ABCGH$ $(CG \rightarrow H$ and $CG \subseteq AGBC)$
  4. $result = ABCGHI$ $(CG \rightarrow I$ and $CG \subseteq AGBCH)$

Step-by-step approach to Find AG^+

1. *result = AG*
2. ***While loop*** *(**1st While loop**)*
3. ***For Loop***

1. take $A \rightarrow B$

$A \subseteq$ result

$result = result \cup B = \{AGB\}$

2. take $A \rightarrow C$

$A \subseteq$ result

$result = result \cup C = \{AGBC\}$

3. Take $CG \rightarrow H$

$CG \subseteq$ result

$result = result \cup H = \{AGBCH\}$

4. Take $CG \rightarrow I$

$CG \subseteq$ result

$result = result \cup I = \{AGBCHI\}$

5. *Take* $B \rightarrow H$

$B \subseteq$ result

$result = result \cup H = \{AGBCHI\}$

End of For Loop

Closure of Attribute Sets

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

Algorithm to **compute** a^+, the **closure** of **a** 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
```

At the end of 1st While loop ***result= { AGBCHI}***
So when 2nd While loop starts result= { AGBCHI}

1. ***While loop*** *(**2nd While loop**)*
2. ***For Loop***
   1. take $A \rightarrow B$
      $A \subseteq$ result
      $result = result \cup B$ ==*{ AGBCHI}*
   2. take $A \rightarrow C$
      $A \subseteq$ result
      $result = result \cup C$ ==*{ AGBCHI}*
   3. Take $CG \rightarrow H$
      $CG \subseteq$ result
      $result = result \cup H$ =*{ AGBCHI}*
   4. Take $CG \rightarrow I$
      $CG \subseteq$ result
      $result = result \cup I$ =*{ AGBCHI}*
   5. *Take* $B \rightarrow H$
      $B \subseteq$ result
      $result = result \cup H$ =*{ AGBCHI}*

End of For Loop
Result from While loop 1 and While loop 2 are same *, there is* ***no change in result*** *, therefore* ***exit While loop***

AG^+= {AGBCHI } AG $\rightarrow$ R

Exercises

1. *R (A, B, C, D)*

   $F = \{ A \rightarrow B \ , B \rightarrow C \ , \quad B \rightarrow D \ \}$, Find A^+ , B^+

2. *R (A,B,C,D)*

   $F = \{C \rightarrow D \ , A \rightarrow B \ , B \rightarrow C \ \}$ Find A^+ , C^+

1.

$A^+ = \{A, B, C, D\}$

$B^+ = \{ B, C, D \}$

2.

$A^+ = \{A, B, C, D\}$

$C^+ = \{C, D \}$

Uses of Attribute Closure

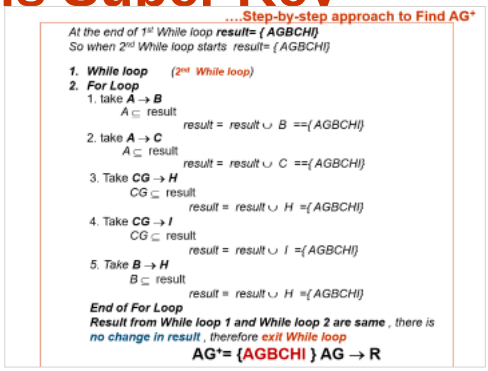
There are several uses of the attribute closure algorithm:

1. **Testing for superkey:**
   - To test if α is a superkey, we compute α^+, and check if α^+ contains all attributes of R. i.e $\alpha \rightarrow \mathbf{R}$
2. **Testing functional dependencies**
   - To check if a functional dependency $\alpha \rightarrow \beta$ holds (or, in other words, is it in F^+), just **check if $\beta \subseteq \alpha^+$.**
   - That is, we compute α^+ by using attribute closure, and then check if it contains β.
   - Is a simple and cheap test, and very useful
3. **Computing closure of F i.e. F^+**
   - For each $\gamma \subseteq \boldsymbol{R}$, we find the closure γ^+, and for each $\boldsymbol{S} \subseteq \gamma^+$, we output a functional dependency $\gamma \rightarrow \boldsymbol{S}$.

1. Example Testing for superkey & Candidate Key:

- $R = (A, B, C, G, H, I)$
- $F = \{ A \rightarrow B$
  $A \rightarrow C$
  $CG \rightarrow H$
  $CG \rightarrow I$
  $B \rightarrow H \ \}$
- **Is AG** a Super Key ? *
- **Is *AG*** a candidate key**?** *
  1. **Is AG a super key?**
     i. Does $AG \rightarrow R$ *?* == Is $(AG)^+ \supseteq R$
  2. **Is any subset of AG a super key?**
     i. Does $A \rightarrow R$? == Is $(A)^+$ contains R
     ii. Does $G \rightarrow R$? == Is $(G)^+$ contains R
- Is **A** is super key?

Step-by-step approach to Check AG is Super Key



Is AG is Super Key?

- Find **AG^+**
- we know AG^+ is **$AG^+ = \{ABCGHI\}$**
- i.e. $R \subseteq (AG)^+$
  - Means R ={A,B,C,G,H,I }, all attributes of R Contained in **AG^+**
- Therefore **$AG \rightarrow R$**
- Therefore **AG is Super key**

***Reference**: slide 21*

- *K* is a super key for relation schema *R* if and only if $K \rightarrow R$
- *K* is a candidate key (minimal Super Key) for *R* if and only if
  - $K \rightarrow R$ *(means K is Super key)*, and
  - for no $\alpha \subset K$, $\alpha \rightarrow R$

Step-by-step approach to Check AG is Candidate Key

***Reference**: slide 21*

- *K* is a super key for relation schema *R* if and only if $K \rightarrow R$
- *K* is a candidate key (minimal Super Key) for *R* if and only if
  - $K \rightarrow R$ *(means K is Super key)*, and
  - for no $\alpha \subset K$, $\alpha \rightarrow R$

IS AG Candidate key ?

We know AG is Super key.

Check further, is there any subset of AG is Super Key?

Subsets of AG are A & G

Check is $A \rightarrow R$ or $G \rightarrow R$?

If any one subset is Super Key then AG is not Candidate key, but only super key.

If **none of subsets** are **Super key** then **AG** is **Candidate Key**.

Find A^+ & G^+

$A^+ = \{ A,B,C,H \}$ **A** $\not\rightarrow$ **R** , Therefore **A** is **Not Super Key**

$G^+ = \{ G \}$ **G** $\not\rightarrow$ **R** , Therefore **G** is **Not Super Key**

None of subsets of AG are Super Keys –

Therefore **AG is Candidate Key (Minimal Super Key)**

2.Example for Testing functional dependencies

To check if a functional dependency $\alpha \rightarrow \beta$ holds (or, in other words, is it in F^+), just check if $\beta \subseteq \alpha^+$. If $\beta \notin \alpha^+$ then $\alpha \not\rightarrow \beta$

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

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

Is $CG \rightarrow A$ functional dependency holds ?

Solution:

- Find CG^+
- IF $A \in CG^+$ Then

  $CG \rightarrow A$ holds

  ELSE

  $CG \not\rightarrow A$ (means functional dependency do not hold)

Exercise:

Is $AC \rightarrow H$ holds ?	**YES**
Is $AG \rightarrow I$ holds ?	**YES**
Is $AI \rightarrow G$ holds ?	**NO**

Exercises

- Consider a relation
  - gradeInfo (rollNo, studName, course, grade)

- Suppose the following FD^s hold:
  - rollNo, course →grade;
  - studName, course →grade ;
  - rollNo → studName;
  - studName → rollNo

- Is this a candidate Key (studName, course) ?

Exercises

- Consider a Schema R = (A, B, C, D, E)
- FDs F = {A →B, B →C, C →A, AB →E, BD →A}
- IS AC →B holds ?
- IS C → E holds ?
- IS AE → D holds ?

3. *Example for Computing F+

For each $\gamma \subseteq R$, we find the closure γ^+, and for each $S \subseteq \gamma^+$, we output a functional dependency $\gamma \rightarrow S$.

$R = (A, B, C)$

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

All the **subsets of R** are –A,B,C,AB,AC,BC,ABC

Take one subset from above **set say -A** and **find A+**

A+ ={A ,B,C}

Subsets of A+ *are* - A,B,C,AB,AC,BC,ABC

Therefore following functional dependencies hold

$A \rightarrow B$, $A \rightarrow C$, $A \rightarrow AB$, $A \rightarrow AC$, $A \rightarrow BC$, $A \rightarrow ABC$. ----(1)

Similarly , **take next subset say B** and **Find B+**

B+={B,C}

Subsets of B+ *are* - B,C,BC

Therefore following functional dependencies hold

B →C ,B →BC ----(2)

Take subset C , Find C+

C+ ={ C}

Therefore **C →C (similarly A →A and B →B holds)** holds ----(3)

….Example for Computing F^+

Take AB

$$\mathbf{A\,B^+ = \{A, B, C\}}$$

Subsets of AB+ are - A , B , C , A B , A C , B C , A B C

Therefore following functional dependencies hold

$$\mathbf{AB \rightarrow B\ , AB \rightarrow C\ , A\,B \rightarrow AB,\ AB \rightarrow AC,\ A\,B \rightarrow BC,\ AB \rightarrow ABC}$$ ----(4)

Take BC

$$\mathbf{B\,C^+ = \{\ B, C\}}$$

Subsets of BC+ are - B , C , B C

Therefore following functional dependencies hold

$$\mathbf{BC \rightarrow B, BC \rightarrow C, BC \rightarrow BC}$$ ---- (5)

Take AC

$$\mathbf{A\,C^+ = \{A, B, C\}}$$

Subsets of AC^+ are - A , B , C , A B , A C , B C , A B C

Therefore following functional dependencies hold

$$\mathbf{AC \rightarrow B\ , AC \rightarrow C\ , AC \rightarrow AB,\ AC \rightarrow AC,\ AC \rightarrow BC,\ AC \rightarrow ABC}.$$ ----(6)

….Example for Computing F^+

Take ABC

ABC+={A , B , C }

Subsets of ABC^+ are - A , B , C , A B , A C , B C , A B C

Therefore following functional dependencies hold

ABC → B , ABC → C , ABC→ AB, ABC → AC, ABC→ BC, ABC → ABC. ----(7)

From (1) (2) (3) (4) (5) (6) (7)

by eliminating duplicates we can get all functional dependencies set

$\mathbf{F^+}$ = { $A \to A$, $B \to B$, $C \to C$, $A \to B$, $A \to C$, $A \to AB$, $A \to AC$, $A \to BC$,
$A \to ABC$, $B \to C$,$B \to BC$, $AB \to B$, $AB \to C$, $A B \to AB$, $AB \to AC$,
$A B \to BC$, $AB \to ABC$, $AC \to B$, $AC \to C$, $AC \to AB$, $AC \to AC$,
$AC \to BC$, $AC \to ABC$, $ABC \to B$, $ABC \to C$, $ABC \to AB$, $ABC \to AC$,
$ABC \to BC$, $ABC \to ABC$ }

Canonical Cover

- Sets of functional dependencies may have redundant dependencies that can be inferred from the others
  - For example: $A \rightarrow C$ is redundant in: $\{A \rightarrow B,\ \ B \rightarrow C,\ A \rightarrow C\}$
  - Parts of a functional dependency may be redundant
    - E.g.: on RHS: $\{A \rightarrow B,\ \ B \rightarrow C,\ \ A \rightarrow CD\}$ can be simplified to

      $$\{A \rightarrow B,\ \ B \rightarrow C,\ \ A \rightarrow D\}$$

    - E.g.: on LHS: $\{A \rightarrow B,\ \ B \rightarrow C,\ \ AC \rightarrow D\}$ can be simplified to

      $$\{A \rightarrow B,\ \ B \rightarrow C,\ \ A \rightarrow D\}$$

- Intuitively, a canonical cover of F is a "minimal" set of functional dependencies equivalent to F, **having no redundant dependencies** or redundant parts of dependencies

Extraneous Attributes

An attribute of a functional dependency is said to be **extraneous** if we can remove it without changing the closure of the set of functional dependencies.

Formal Definition for **extraneous Attributes.**

- Consider a set ***F*** of functional dependencies and the functional dependency $\alpha \rightarrow \beta$ in ***F***.
  - Attribute **A** is **extraneous** in α if $A \in \alpha$ (an attribute A on the left side of $\alpha \rightarrow \beta$)
    and *F* logically implies $(F - \{\alpha \rightarrow \beta\}) \cup \{(\alpha - A) \rightarrow \beta\}$.
  - Attribute ***A*** is **extraneous** in β if $A \in \beta$ (an attribute **A** on the right side of $\alpha \rightarrow \beta$) and the set of functional dependencies $(F - \{\alpha \rightarrow \beta\}) \cup \{\alpha \rightarrow (\beta - A)\}$ logically implies *F*.

Extraneous Attributes

- Example: Given $F = \{A \rightarrow C, AB \rightarrow C\}$
  - B is extraneous in $AB \rightarrow C$
  - Means B is *left Extraneous* in $AB \rightarrow C$, therefore $AB \rightarrow C$ can be replaced *by* $A \rightarrow C$

- Example: Given $F = \{A \rightarrow C, AB \rightarrow CD\}$
  - C is extraneous in $AB \rightarrow CD$
  - Means C is *right extraneous in* $AB \rightarrow CD$, therefore $AB \rightarrow CD$ can be replaced by $AB \rightarrow D$

Testing if an Attribute is Extraneous

- Consider a set *F* of functional dependencies and the functional dependency $\alpha \rightarrow \beta$ in *F*.

- **To test if attribute $A \in \alpha$ is extraneous in α (Left extraneous)**
  1. compute $(\{\alpha\} - A)^+$ using the dependencies in *F*
  2. check that **$(\{\alpha\} - A)^+$ contains β**; if it does, *A* is extraneous in α

- **To test if attribute $A \in \beta$ is extraneous in β (Right extraneous)**
  1. compute α^+ using only the dependencies in F'

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

  2. check that α^+ **contains *A***; if it does, *A* is extraneous in β

Canonical Cover

- A **canonical cover** for *F* is a set of dependencies F_c such that
  - ***F*** logically implies all dependencies in $\boldsymbol{F_c}$, and
  - $\boldsymbol{F_c}$ logically implies all dependencies in ***F***, and
  - No functional dependency in $\boldsymbol{F_c}$ contains an extraneous attribute, and
  - Each left side of functional dependency in $\boldsymbol{F_c}$ is unique.

Canonical Cover: Algorithm

- To compute a canonical cover for *F*:

- $F_c = F$

  repeat

  Use the union rule to replace any dependencies in F_c

  $\alpha_1 \rightarrow \beta_1$ and $\alpha_1 \rightarrow \beta_2$ with $\alpha_1 \rightarrow \beta_1 \beta_2$

  Find a functional dependency $\alpha \rightarrow \beta$ in F_c with an extraneous attribute either in α or in β

  /* Note: test for extraneous attributes done **using F_c,** not F */

  If an extraneous attribute is found, delete it from $\alpha \rightarrow \beta$

  until *F* does not change

- Note: Union rule may become applicable after some extraneous attributes have been deleted, so it has to be re-applied

Computing a Canonical Cover

- $R = (A, B, C)$
  $F = \{A \rightarrow BC,$
  $B \rightarrow C,$
  $A \rightarrow B,$
  $AB \rightarrow C\}$
- Combine $A \rightarrow BC$ and $A \rightarrow B$ into $A \rightarrow BC$
  - Set is now $\{A \rightarrow BC, B \rightarrow C, AB \rightarrow C\}$ **Applying UNION Rule.**
- ***A* is extraneous** in $AB \rightarrow C$ **Checking Left extraneous**
  - Check if the result **of deleting A** from $\boldsymbol{AB \rightarrow C}$ is implied by the other dependencies
    - **Yes**: in fact, $B \rightarrow C$ is already present!
  - Set is now $\boldsymbol{\{A \rightarrow BC, B \rightarrow C\}}$
- ***C* is extraneous** in $A \rightarrow BC$ **Checking Right extraneous**
  - Check if $A \rightarrow C$ is logically implied by $A \rightarrow B$ and the other dependencies
    - **Yes***:* using transitivity on $A \rightarrow B$ *and* $B \rightarrow$ C.
      - Can use attribute closure of *A* in more complex cases
- The canonical cover is:

$$\mathbf{F_c = \{ A \rightarrow B ,\ B \rightarrow C \}}$$

Exercise-1

- Consider a Schema R = (A, B, C, D, E)
- FDs F ={ A →BCD, BC →DE, B →D, D →A}
- Find Cannonical cover.

To test if attribute $A \in \alpha$ is extraneous in α (Left extraneous)

1. compute $(\{\alpha\} - A)^+$ using the dependencies in *F*
2. check that **$(\{\alpha\} - A)^+$ contains β**; if it does, *A* is extraneous in α

To test if attribute $A \in \beta$ is extraneous in β (Right extraneous)

1. compute α^+ using only the dependencies in F′

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

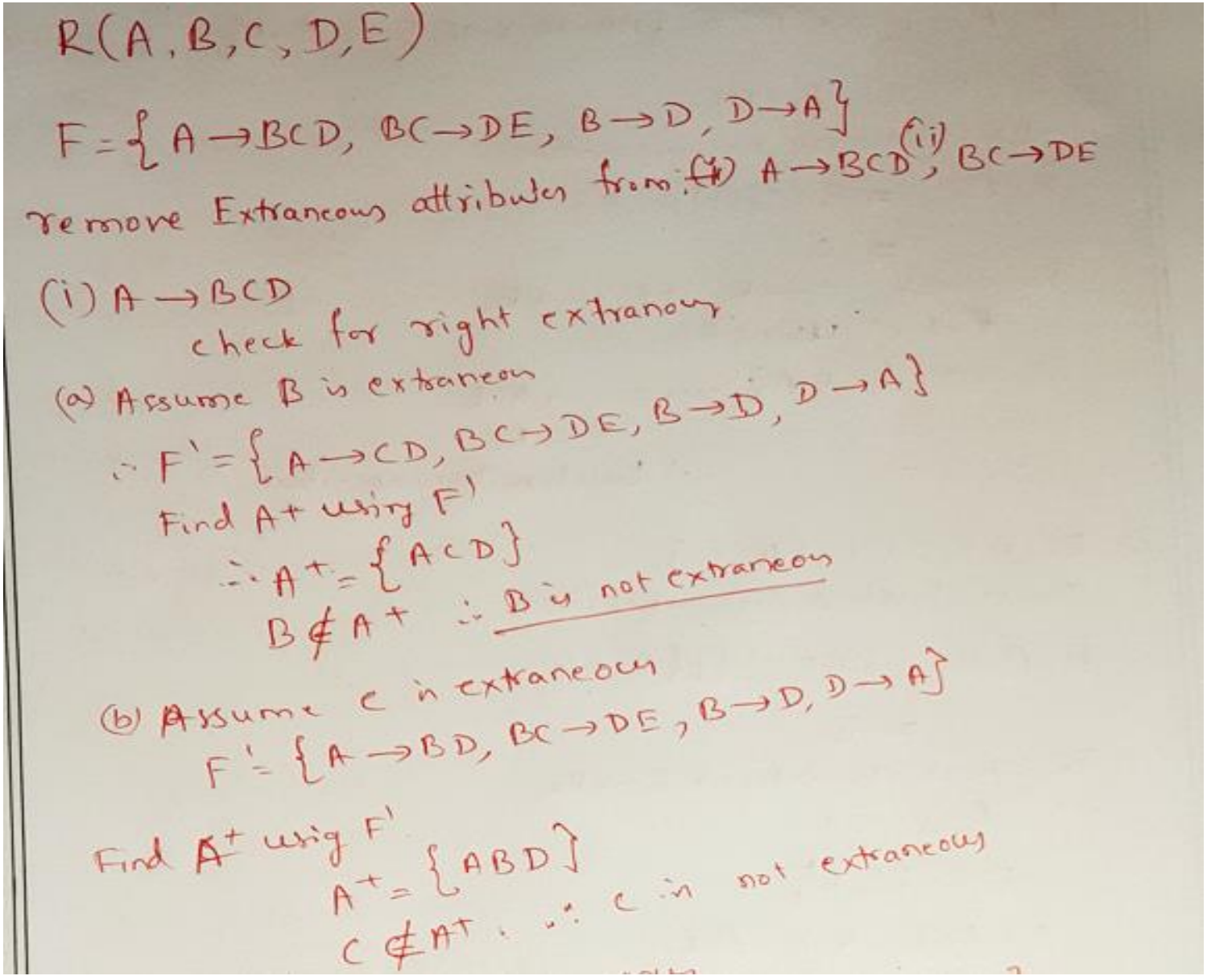
2. check that α^+ **contains *A***; if it does, *A* is extraneous in β

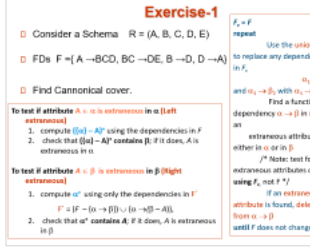
$F_c = F$
repeat
Use the union rule to replace any dependencies in F_c
$\alpha_1 \rightarrow \beta_1$ and $\alpha_1 \rightarrow \beta_2$ with $\alpha_1 \rightarrow \beta_1 \beta_2$
Find a functional dependency $\alpha \rightarrow \beta$ in F_c with an extraneous attribute either in α or in β
/* Note: test for extraneous attributes done **using F_c**, not F */
If an extraneous attribute is found, delete it from $\alpha \rightarrow \beta$
until *F* does not change

Consider a Schema R = (A, B, C, D, E)

FDs F ={ A →BCD, BC →DE, B →D, D →A}

Find Cannonical cover.



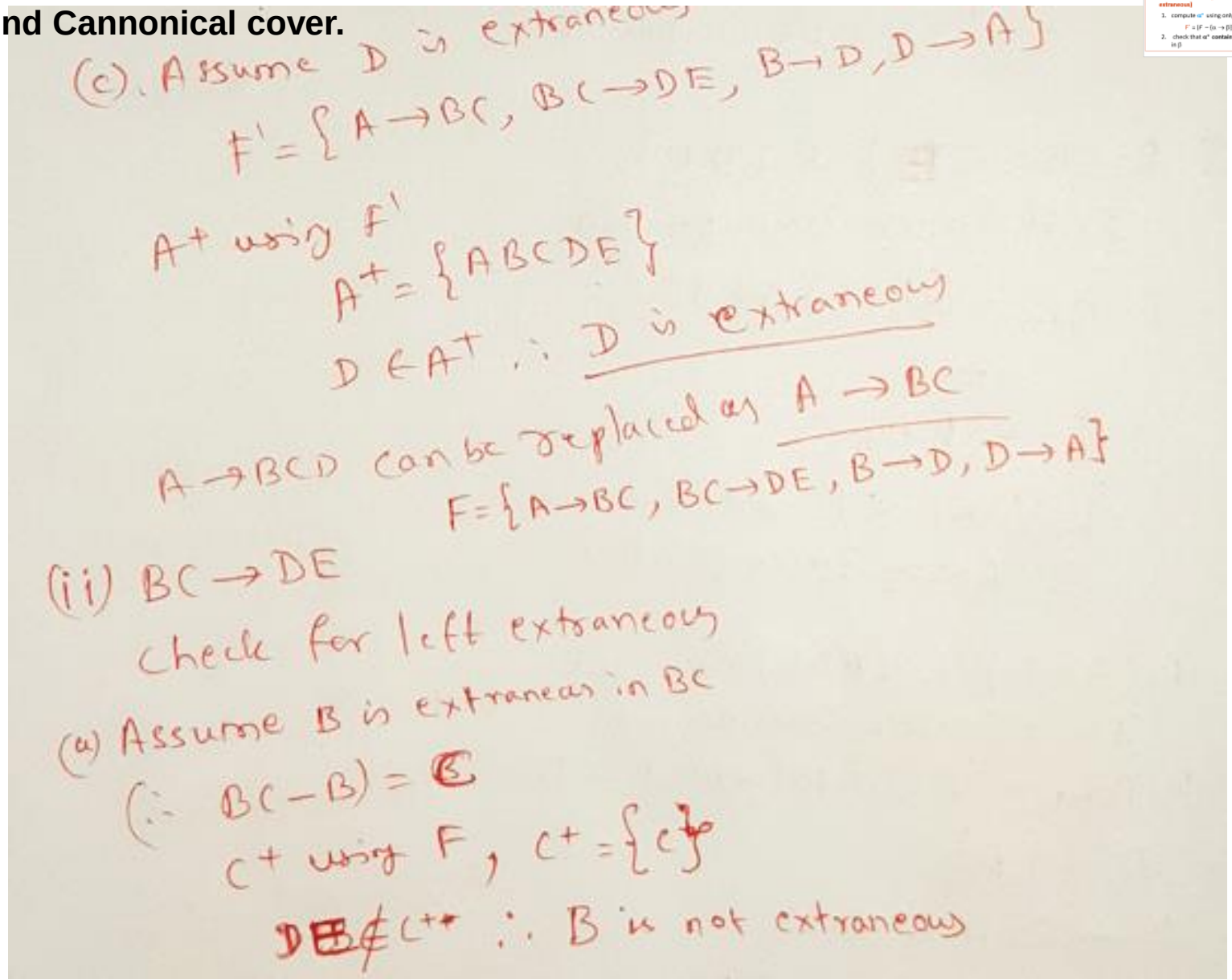


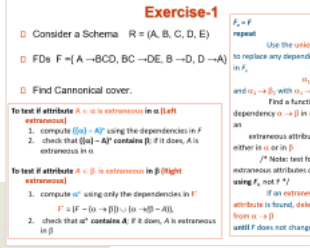
Union Rule can't be applied

Consider a Schema R = (A, B, C, D, E)

FDs F ={ A →BCD, BC →DE, B →D, D →A}

Find Cannonical cover.

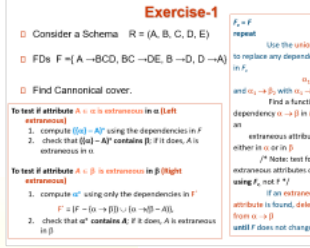




Consider a Schema R = (A, B, C, D, E)

FDs F ={ A →BCD, BC →DE, B →D, D →A}

Find Cannonical cover.



(b) Assume C in extraneous in BC

$\therefore BC - C = B \quad \therefore B^+$ using F

$B^+ = \{BDACDE\}$

$DE \in B^+ \therefore$ C is extraneous

replace $BC \rightarrow DE$ as $B \rightarrow DE$

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

(iii) $B \rightarrow DE$

check for right extraneous

(a) Assume D in extraneous in DE

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

B^+ using F'

$B^+ = \{BEDAC\}$

$D \in B^+ \therefore$ D is extraneous

(b) replace $B \rightarrow DE$ using $B \rightarrow E$

$F_c = \{A \rightarrow BC, B \rightarrow E, B \rightarrow D, D \rightarrow A\}$

$F_c = \{A \rightarrow BC, B \rightarrow DE, D \rightarrow A\}$

Exercise-2

- Consider a Schema R = (A, B, C)
- FDs F = { A →BC, B →C, A→B, AB →C }
- Find Cannonical cover.

To test if attribute $A \in \alpha$ is extraneous in α (Left extraneous)

1. compute $(\{\alpha\} - A)^+$ using the dependencies in *F*
2. check that **$(\{\alpha\} - A)^+$ contains β**; if it does, *A* is extraneous in α

To test if attribute $A \in \beta$ is extraneous in β (Right extraneous)

1. compute α^+ using only the dependencies in F′

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

2. check that **α^+ contains *A*;** if it does, *A* is extraneous in β

$F_c = F$

repeat

Use the union rule to replace any dependencies in F_c

$\alpha_1 \rightarrow \beta_1$ and $\alpha_1 \rightarrow \beta_2$ with $\alpha_1 \rightarrow \beta_1 \beta_2$

Find a functional dependency $\alpha \rightarrow \beta$ in F_c with an extraneous attribute either in α or in β

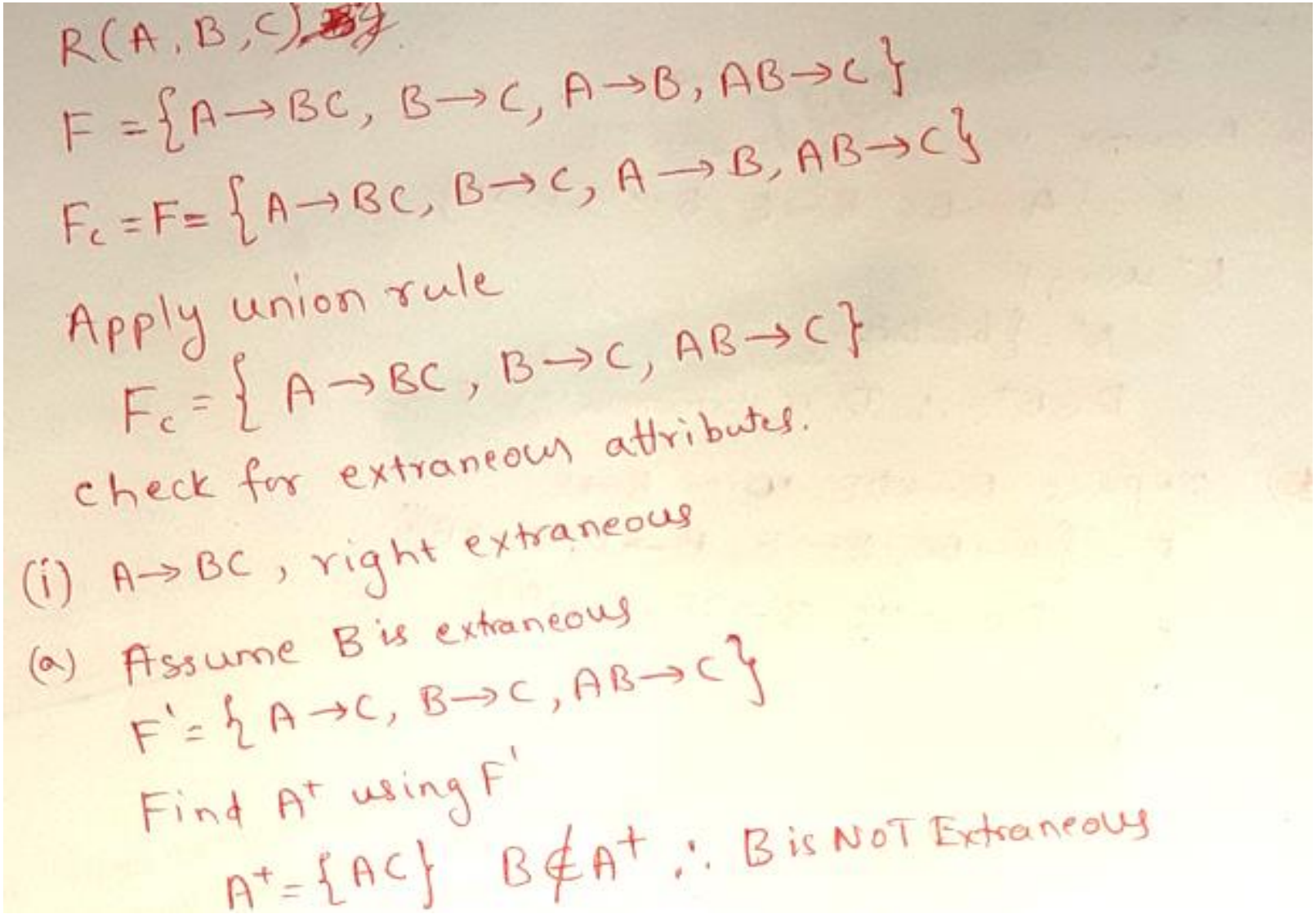
/* Note: test for extraneous attributes done **using F_c,** not F */

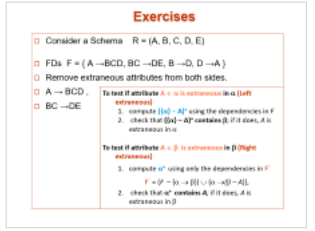
If an extraneous attribute is found, delete it from $\alpha \rightarrow \beta$

until *F* does not change

Consider a Schema R = (A, B, C)

FDs F = { A →BC, B →C, A→B, AB →C }
Find Cannonical cover

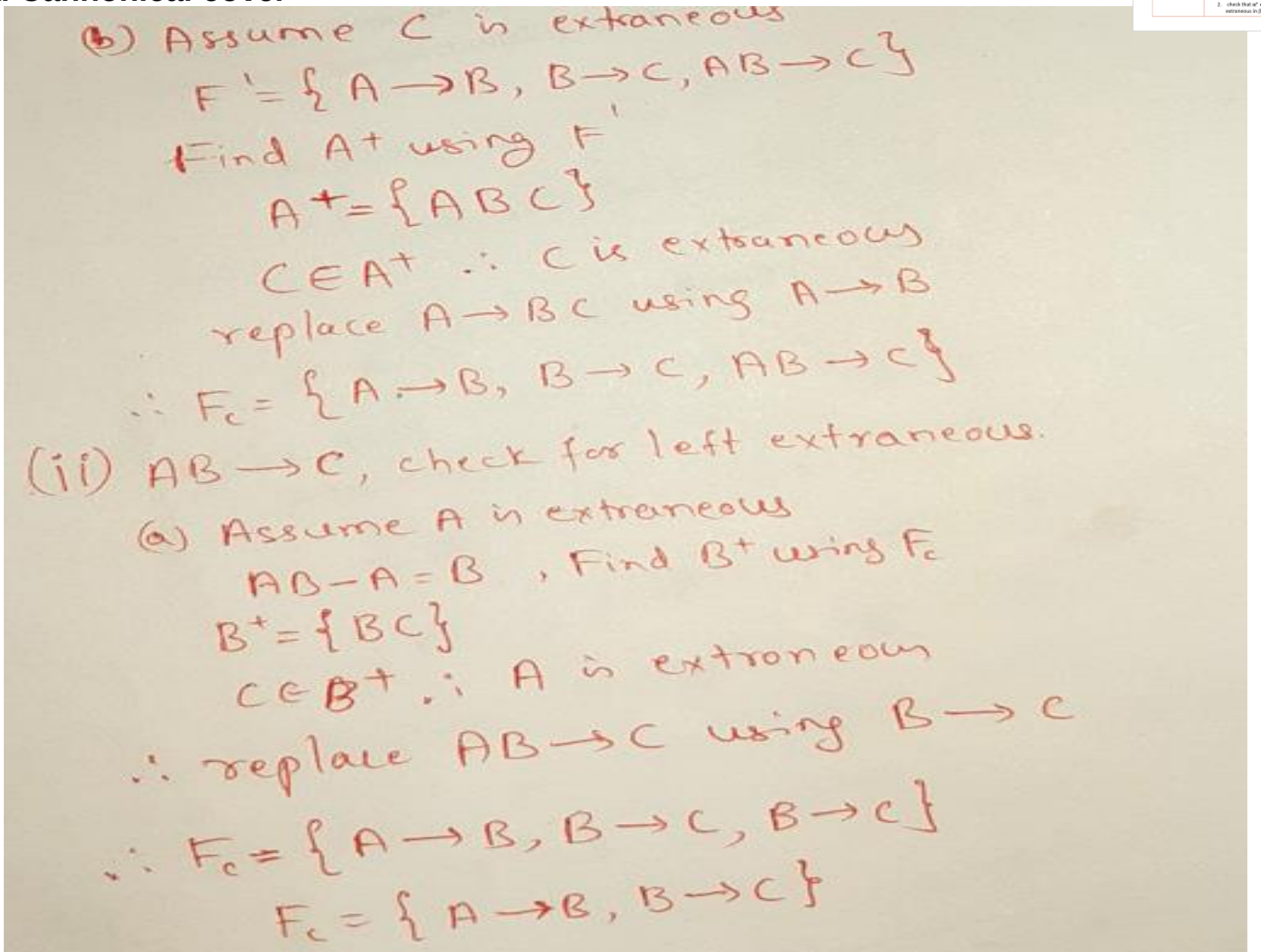




Consider a Schema R = (A, B, C)

FDs F = { A →BC, B →C, A→B, AB →C }

Find Cannonical cover



Exercises

- Consider a Schema R = (A, B, C, D, E)
- FDs F = { A →BCD, BC →DE, B →D, D →A }
- Remove extraneous attributes from both sides.
- A → BCD ,
- BC →DE

To test if attribute A ∈ α is extraneous in α (Left extraneous)
1. compute ((α) − A)+ using the dependencies in F
2. check that ((α) − A)+ contains β; if it does, A is extraneous in α

To test if attribute A ∈ β is extraneous in β (Right extraneous)
1. compute α+ using only the dependencies in F' = (F − {α → β}) ∪ {α → (α − A)}.
2. check that α+ contains A; if it does, A is extraneous in β

Lossless-join Decomposition

- For the case of **R = (R_1, R_2)**, we require that for all possible relations *r* on schema *R*

$$r = \Pi_{R1}(r) \bowtie \Pi_{R2}(r)$$

- A decomposition of *R* into R_1 and R_2 is lossless join if at least one of the following dependencies is in F^+:
  - $R_1 \cap R_2 \rightarrow R_1$
  - $R_1 \cap R_2 \rightarrow R_2$

This implies common attribute of R1 and R2 must be super key of R_1 or R_2

- The above functional dependencies are a sufficient condition for lossless join decomposition; the dependencies are a necessary condition only if all constraints are functional dependencies

Example

- $R = (A, B, C)$
  $F = \{A \rightarrow B, B \rightarrow C)$
  - Can be decomposed in two different ways
- $\boldsymbol{R_1 = (A, B),\quad R_2 = (B, C)}$
  - Lossless-join decomposition:

    $$\boldsymbol{R_1 \cap R_2 = \{B\} \text{ and } B \rightarrow BC \quad i.e.\ B \rightarrow R_2}$$

  - Dependency preserving
- $\boldsymbol{R_1 = (A, B),\quad R_2 = (A, C)}$
  - Lossless-join decomposition:

    $$\boldsymbol{R_1 \cap R_2 = \{A\} \text{ and } A \rightarrow AB}$$

  - Not dependency preserving
    (cannot check $B \rightarrow C$ without computing $R_1 \bowtie R_2$)

Decomposition: EXAMPLES

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

F = {AB→C, C→E, B→D, E→A}

Assume R is Decomposed into R_1 and R_2 as below-

- R_1(B,C,D) R_2(A,C,E)

Is this a Lossless Decomposition ?Check it.

- R_1(B,E,D) R_2(A,C,E)

Is this a Lossless Decomposition ?Check it.

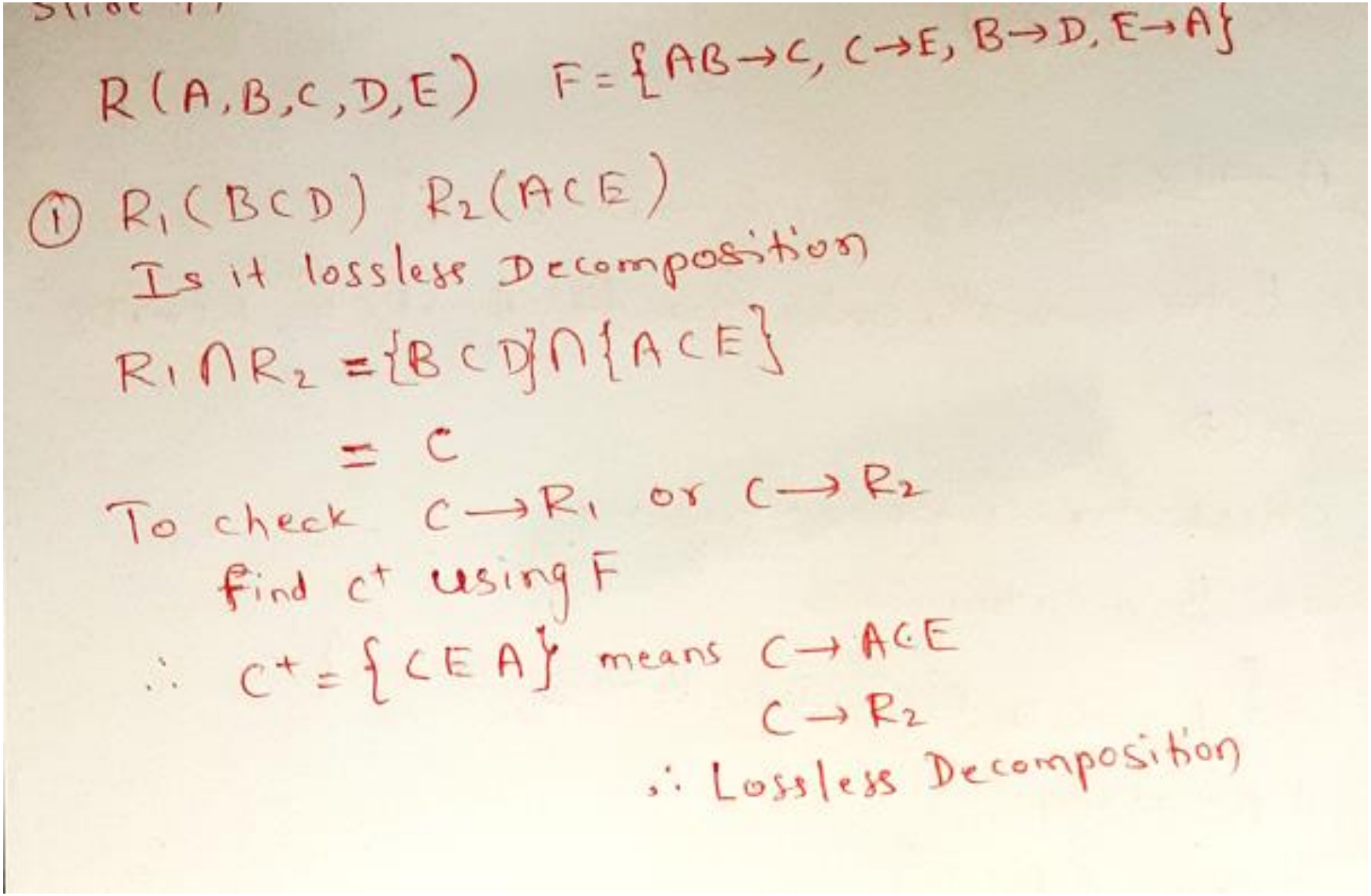
- R1(A,B,E,C) R2(D,E)

Is this a Lossless Decomposition ?Check it.

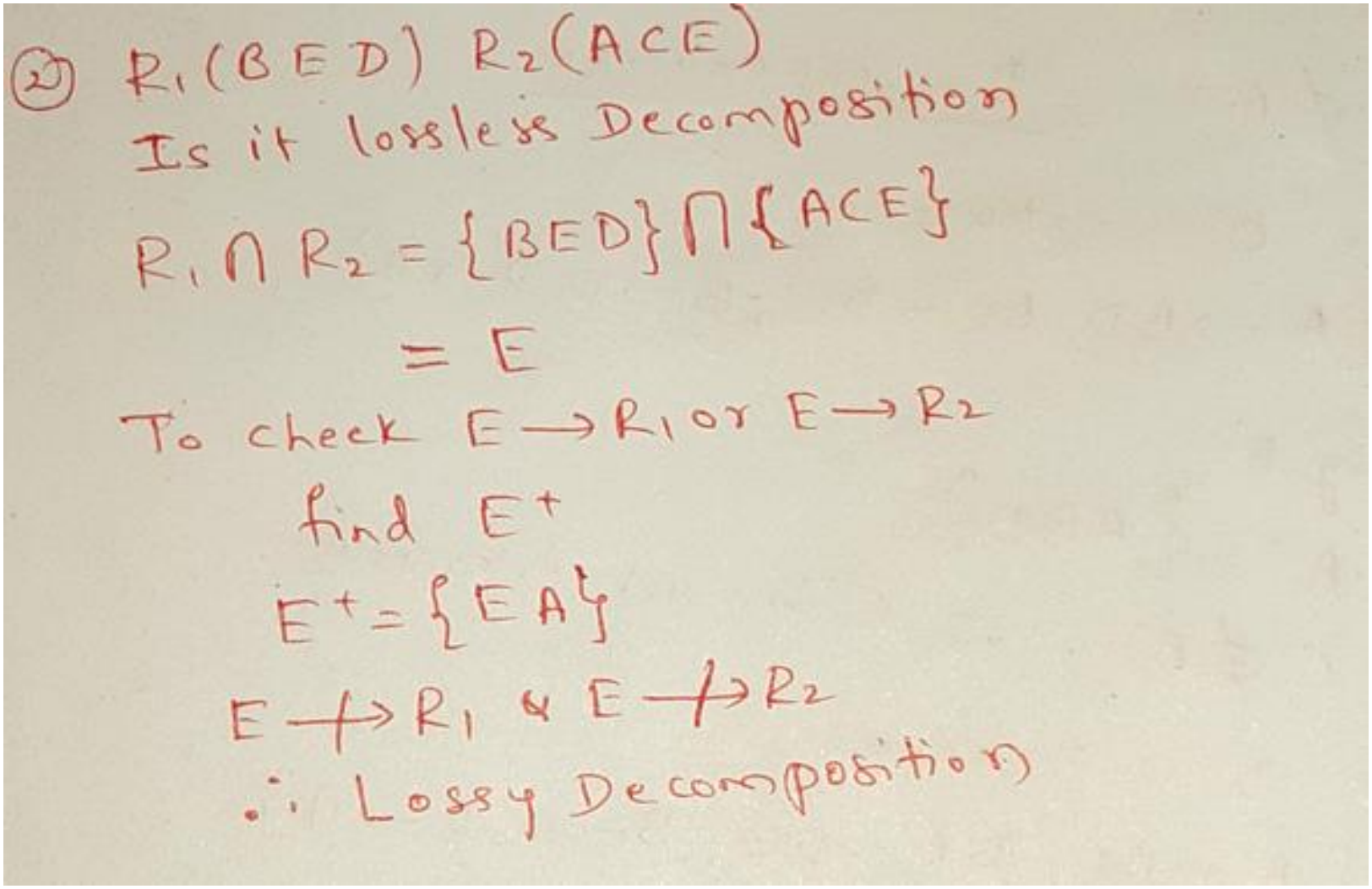
- R1(B,C,D) R2(A,D,E)

Is this a Lossless Decomposition ?Check it.

$R_1(B,C,D)$ $R_2(A,C,E)$
Is this a Lossless
Decomposition ?Check it.



R_1(B,E,D) R_2(A,C,E)
Is this a Lossless Decomposition ?Check it.



Dependency Preservation

- Let F_i be the set of dependencies F^+ that include only attributes in R_i.
  - A decomposition is **dependency preserving**, if

    $$(F_1 \cup F_2 \cup \ldots \cup F_n)^+ = F^+$$

  - If it is not, then checking updates for violation of functional dependencies may require computing joins, which is expensive.

Testing for Dependency Preservation

- To check if a dependency $\alpha \rightarrow \beta$ is preserved in a decomposition of R into $R_1, R_2, \ldots, R_n$ we apply the following test (with attribute closure done with respect to F)

  result = α
  while (changes to *result*) do
    for each R_i in the decomposition
      $t = (result \cap R_i)^+ \cap R_i$
      $result = result \cup t$

  - If ***result* contains all attributes in β**, then the functional dependency $\alpha \rightarrow \beta$ **is preserved**.

- **We apply the test on all dependencies in F to check if a decomposition is dependency preserving**

- This procedure **takes polynomial time**, instead of the exponential time required to compute F^+ **and** $(F_1 \cup F_2 \cup \ldots \cup F_n)^+$

Dependency Preservation-Example

- Given the following:

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

  F = {AB→C, C→E, B→D, E→A}

  Assume R is Decomposed into R_1 and R_2 as below-

  R_1(B,C,D) R_2(A,C,E)

- **Is this decomposition dependency preserving?**

Example continued…

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

Decomposition: R1(B,C,D) R2(A,C,E)

Is AB→C Preserved?....

Result=AB

For Result ∩ R1 = AB ∩ BCD = B

$\{B\}^+ = BD$

$\{B\}^+ \cap R1 = BD \cap BCD = BD$

Update Result = AB ∪ BD = ABD,

continue to next for loop

result = α

while (changes to *result*) do

for each R_i in the decomposition

$t = (result \cap R_i)^+ \cap R_i$

$result = result \cup t$

Example continued…

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

Decomposition: R1(B,C,D) R2(A,C,E)

Is AB→C Preserved? ……

Result=ABD

For Result ∩ R2 = ABD ∩ ACE = A

$\{A\}^+ = A$

$\{A\}^+ \cap R2 = A \cap ACE = A$

Update Result, Result is still ABD

Since Result changed,

continue to next while loop

result = α

while (changes to *result*) do

for each R_i in the decomposition

$t = (result \cap R_i)^+ \cap R_i$

$result = result \cup t$

Example continued…

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

Decomposition: R1(B,C,D) R2(A,C,E)

Is AB→C Preserved? ……

Result=ABD

For Result ∩ R1 = ABD ∩ BCD = BD

$\{BD\}^+ = BD$

$\{BD\}^+ \cap R1 = BD \cap BCD = BD$

Update Result = ABD ∪ BD = ABD,

Continue to next for loop

result = α

while (changes to *result*) do

for each R_i in the decomposition

$t = (result \cap R_i)^+ \cap R_i$

$result = result \cup t$

Example

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

Decomposition: R1(B,C,D) R2(A,C,E)

List the functional dependencies which are not preserved?

Is AB→C Preserved?

Result=ABD

For Result ∩ R2 = ABD ∩ ACE = A

$\{A\}^+ = A$

$\{A\}^+ \cap R2 = A \cap ACE = A$

Update Result, Result is still ABD

Since Result hasn't changed, we stop here.

Check, Is Result Contains C ? **NO**

Therefore we can conclude

AB→C is not preserved.

Similarly check other functional dependencies.

Example continued…

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

Decomposition: R1(B,C,D) R2(A,C,E)

Is B→D Preserved ?.....

Result=B

For Result ∩ R1 = B ∩ BCD = B

$\{B\}^+ = BD$

$\{B\}^+ \cap R1 = BD \cap BCD = BD$

Update Result = B ∪ BD = BD

Continue to next for loop

result = α

while (changes to *result*) do

for each R_i in the decomposition

$t = (result \cap R_i)^+ \cap R_i$

$result = result \cup t$

Example continued…

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

Decomposition: R1(B,C,D) R2(A,C,E)

Is B→D Preserved ?

Result=BD

For Result ∩ R2 = BD ∩ ACE = Φ

$\{\Phi\}^+ = \Phi$

$\{\Phi\}^+ \cap R2 = \Phi \cap ACE = \Phi$

Update Result = B ∪ Φ = BD

Continue to next while loop.

Result unchanged. Stop

Since D belongs to Result,

B→D is preserved.

result = α

while (changes to *result*) do

for each R_i in the decomposition

$t = (result \cap R_i)^+ \cap R_i$

$result = result \cup t$

Example continued…

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

Decomposition: R1(B,C,D) R2(A,C,E)

Is C→E Preserved ?....

Result=C

For Result $\cap$ R1 = C $\cap$ BCD = C

$\{C\}^+ = CEA$

$\{C\}^+ \cap R1 = CEA \cap BCD = C$

Update Result = C $\cup$C= C

Continue to next for loop

result = α

while (changes to *result*) do

for each R_i in the decomposition

$t = (result \cap R_i)^+ \cap R_i$

$result = result \cup t$

Example continued…

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

Decomposition: R1(B,C,D) R2(A,C,E)

Is C→E Preserved ?

Result=C

For Result ∩ R2 = C ∩ ACE = C

$\{C\}^+ = CEA$

$\{C\}^+ \cap R2 = CEA \cap ACE = ACE$

Update Result = C ∪ACE= ACE

In next while loop result remains unchanged hence stop.

Since E is in Result i.e. ACE,

C→E is preserved.

Example continued…

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

Decomposition: R1(B,C,D) R2(A,C,E)

Is E→A Preserved ?.....

Result=E

For Result $\cap$ R1 = E $\cap$ BCD = Φ $\{\Phi\}^+ = \Phi$

$\{\Phi\}^+ \cap R1 = \Phi \cap BCD = \Phi$

Update Result = E $\cup$ Φ = E

Continue to next for loop

Example continued…

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

Decomposition: R1(B,C,D) R2(A,C,E)

Is E→A Preserved ?

Result=E

For Result ∩ R2 = E ∩ ACE = E

$\{E\}^+ = EA$

$\{E\}^+ \cap R1 = EA \cap ACE = EA$

Update Result = E ∪ EA= EA

In next while loop result remains unchanged hence stop.

Since A belongs to Result i.e. EA,

E→A is preserved.

Example continued…

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

Decomposition: R1(B,C,D) R2(A,C,E)

AB→C is not preserved.

C→E is preserved.

B→D is preserved

E→A is preserved.

As one of the functional dependencies AB→C is not is not preserved, Therefore decomposition R1,R2 is not a dependency preserving decomposition.

Normalization

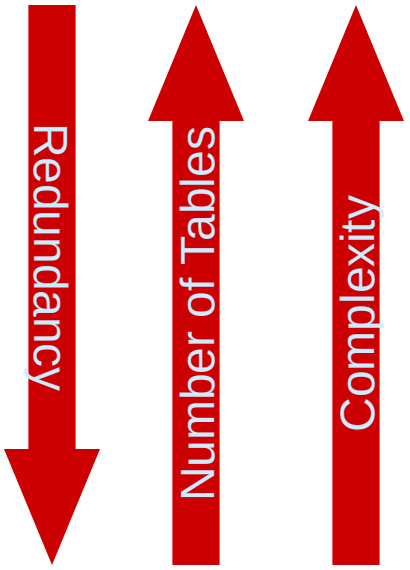
- Normalization is the process of modifying a relation schema based on its Functional Dependencies and primary keys so that it meets certain rules called normal forms.
- It is conducted by evaluating a relation schema to check whether it satisfies particular rules and if not, then decomposing the schema into set of smaller relation schemas that do not violate those rules
- Main purpose is to eliminate data redundancy and avoid data update anomalies.
- A relation is said to be in a particular normal form if it **satisfies certain specified constraints**
- Each of the normal form is **stricter than its predecessors.**

Normalization

- A properly normalized database should have the following characteristics
  - **Atomic** values in each fields
  - Absence of **redundancy.**
  - Minimal use of **null values**.
  - Minimal **loss of information**.

Levels of Normalization

- Levels of normalization based on the amount of redundancy in the database.
- Various levels of normalization are:
  - First Normal Form (1NF)
  - Second Normal Form (2NF)
  - Third Normal Form (3NF)
  - Boyce-Codd Normal Form (BCNF)
  - Fourth Normal Form (4NF)
  - Fifth Normal Form (5NF)
  - Domain Key Normal Form (DKNF)



Most databases should be 3NF or BCNF in order to avoid the database anomalies.

First Normal Form

- A relational schema R is in **first normal form** if the domains of all attributes of R are atomic
- Domain is **atomic** if its elements are considered to be indivisible units
  - Examples of non-atomic domains:
    - Set of names, composite attributes
    - Identification numbers like "CS101" that can be broken up into parts-Dept & RollNo
- Non-atomic values complicate storage and encourage redundant (repeated) storage of data
  - Example: Set of accounts stored with each customer, and set of owners stored with each account

First Normal Form (Cont'd)

- Atomicity is actually a property of how the elements of the domain are used.
  - Example: Strings would normally be considered indivisible
  - Suppose that students are given roll numbers which are strings of the form *CS0012* or *EE1127*
    - If the first two characters are extracted to find the department, the domain of roll numbers is not atomic.
  - Doing so is a bad idea: leads to encoding of information in application program rather than in the database.

Example

Non-Atomic Values

TABLE_PRODUCT

Product ID	Color	Price
1	red, green	15.99
2	yellow	23.99
3	green	17.50
4	yellow, blue	9.99
5	red	29.99

1NF Decomposition

1. Place all items that appear in the repeating group in a new table
2. Duplicate in the new table the primary key of the table from which the repeating group was extracted or vice versa.
3. Designate a primary key for each new table produced.

1NF Decomposition

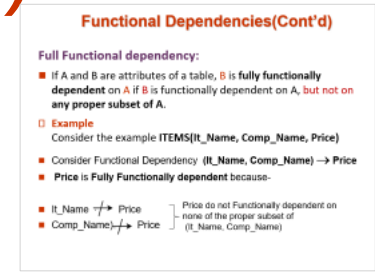
TABLE_PRODUCT_PRICE

Product ID	Price
1	15.99
2	23.99
3	17.50
4	9.99
5	29.99

TABLE_PRODUCT_COLOR

Product ID	Color
1	red
1	green
2	yellow
3	green
4	yellow
4	blue
5	red

Second Normal Form (2NF)



For a table to be in 2NF, there are two requirements

- The database is in first normal form
- All **nonkey** attributes in the table must be fully functionally dependent on the entire primary key

Example (Not 2NF)

Scheme , Book_author(PubId, AuId, Title, Price, AuAddress)

1. **Primary Key is** {PubId, AuId}
2. {PubId, AuID} → {Price}
3. {PubId, AuID} → {Title}
4. {AuID} → {AuAddress}
5. AuAddress is not determined by all key attributes
6. **AuAddress** functionally depends **on AuId** which is a **subset of a key**
7. **Hence AuAddress** partially depends on **Primary Key**

2NF - Decomposition

1. If a data item is fully functionally dependent on only a part of the primary key, move that data item and that part of the primary key to a new table.
2. If other data items are functionally dependent on the same part of the key, place them in the new table also
3. Make the partial primary key copied from the original table the primary key for the new table. Place all items that appear in the repeating group in a new table

Example 1 (Convert to 2NF)

Old Scheme {PubId, AuId, Title ,Price, AuAddress}

New Scheme1 {PubId, AuId, Title, Price}

New Scheme 2 {AuId, AuAddress}

3 NF

A relation schema *R* is in **third normal form** with respect to a set *F* of functional dependencies if, for all functional dependencies in *F*+ of the form $\alpha \rightarrow \beta$, Where $\alpha \subseteq R$ and $\beta \subseteq R$, **at least one** of the following holds:

- $\alpha \rightarrow \beta$ is a **trivial functional dependency**.
- α is a **super key** for *R*.
- Each attribute ***A*** **in** $\beta - \alpha$ is **contained** in a candidate key for *R*.

Note that the third condition above does not say that a single candidate key must contain all the attributes in $\beta - \alpha$; **each attribute A** in $\beta - \alpha$ may be **contained in a different candidate key.**

Any **"schema that satisfies BCNF also satisfies 3NF"**, since each of its functional dependencies would satisfy one of the first two alternatives. **BCNF is therefore a more restrictive** normal form than is 3NF

Testing for 3NF

- Optimization: Need to check only FDs in F, need not check all FDs in F^+.
- Use attribute closure to check for each functional dependency $\alpha \rightarrow \beta$, if **α is a super key.**
- If α is **not a super key**, we have to **verify if each attribute in $\beta - \alpha$ is contained in a candidate key of *R***
  - Attributes in $\beta - \alpha$ may be in different candidate keys of R

3NF Example

- Relation ***dept_advisor***: ***dept_advisor*** (***s_ID, i_ID, dept_name***)

$F = \{(s\_ID, dept\_name) \rightarrow i\_ID,\ i\_ID \rightarrow dept\_name\}$

- Consider that, we have Two candidate keys:
  - (***s_ID, dept_name)***, and ***(i_ID, s_ID)***
- *R* is in 3NF , because
  - In *((s_ID, dept_name* $\rightarrow$ *i_ID*
    - *(s_ID dept_name)* is **a super key (satisfy 2nd Condition)**
  - In *i_ID* $\rightarrow$ *dept_name* ; ***i_ID is not super key*** , *but*
    - *dept_name* is **contained in** a candidate key (s_ID dept_name)
      - **(satisfy 3rd Condition)**

Refer slide for 3NF

3NF Decomposition Algorithm

Let F_c be a canonical cover for F;

$i := 0$;

for each functional dependency $\alpha \rightarrow \beta$ in F_c **do**

begin

$i := i + 1$;

$R_i := \alpha \beta$

end

if none of the schemas R_j, $1 \leq j \leq i$ contains a candidate key for R

then begin

$i := i + 1$;

R_i := any candidate key for R;

end

/* Optionally, remove redundant relations */

repeat

if any schema R_j is contained in another schema R_k

then ***/** delete R_j ***/***

$R_j = R_i$;

$i=i-1$;

return $(R_1, R_2, ..., R_i)$

Note: algorithm ensures:

- each relation schema R_i is in 3NF
- decomposition is dependency preserving and lossless-join

3NF Decomposition example

Assume $R = (A, B, C)$

$F_c = \{A \rightarrow B,\ B \rightarrow C\}$, A is candidate key (if not given, then find yourself).

R is not in 3 NF and Decompose to 3 NF

Solution:

i=0

For loop

take $A \rightarrow B$, i=1 , R_1=A,B

take $B \rightarrow C$, i=2 , R_2 =B,C

IF none of R_1 ,R_2 contains Candidate key (i.e. A) is FALSE

because A is contained in R_1

Repeat

there is neither R_1 contained in R_2 nor R_2 contained in R_1

therefore No Deletion of duplicate schema

Return (R_1,R_2)

"R_1(A,B) & R_2(B,C) are decomposed 3 NF Relations"

3NF Decomposition: An Example

- **Relation schema:**

  cust_banker_branch = (customer_id, employee_id, branch_name, type)

  Assume **(customer_id, employee_id)** is the candidate key*

- The **functional dependencies** for this relation schema are:
  1. customer_id, employee_id → branch_name, type
  2. employee_id → branch_name
  3. customer_id, branch_name → employee_id

- We first **compute a canonical cover**
  - branch_name is extraneous in the r.h.s. of the 1st dependency
  - No other attribute is extraneous, so we get

    F_C = { customer_id, employee_id → type
    employee_id → branch_name
    customer_id, branch_name → employee_id }

3NF Decompsition Example (Cont.)

- The **for** loop generates following 3NF schema:

  (customer_id, employee_id, type)

  (employee_id, branch_name)

  (customer_id, branch_name, employee_id)

  - Observe that **(customer_id, employee_id, type) contains a candidate key of the original schema,** so no further relation schema needs be added

- **At end of for loop**, detect and delete schemas, such as-

  - **(employee_id, branch_name),** which is **subset of** other schemas namely-**(customer_id, branch_name, employee_id**)

  - result will not depend on the order in which FDs are considered

- The resultant simplified 3NF schema is:

  R_1 (customer_id, employee_id, type)

  R_2 (customer_id, branch_name, employee_id)

R(A,B,C,D,E,F,G,H) and F={ ABC →DE , E →BCG, F →AH }

Is it in 3NF? Decompose into 3NF.

- Assume candidate keys are **FE** and **FBC**
- $\mathbf{F_c}$ ={ABC →DE , E →BCG, F →AH }
-

R(A,B,C,D,E,F,G,H,I,J) and

F={ AB → C ,A → DE , B →F, F →GH, D →IJ}

Is it in 3NF? Decompose into 3NF.

- Assume candidate keys are **AB**
- $\mathbf{F_c}$={ AB → C ,A → DE , B →F, F →GH, D →IJ}

Boyce- Codd Normal Form

A relation schema *R* is in BCNF with respect to a set *F* of functional dependencies if for all functional dependencies in F^+ of the form

$$\alpha \rightarrow \beta$$

where $\alpha \subseteq R$ and $\beta \subseteq R$, **at least one of the** following holds:

- $\alpha \rightarrow \beta$ is trivial (i.e., $\beta \subseteq \alpha$)
- α is a super key for *R*

Example schema *not* in BCNF:

instr_dept (*ID, name, salary, dept_name, building, budget*)

Assume F={ *ID* → *name, salary , dept_name, dept_name* → *building, budget* }

instr_dept is not in BCNF because *dept_name* → (*building, budget)*

BECAUSE, ***dept_name*** **is not a superkey**

Decomposing a Schema into BCNF

- Suppose we have a schema *R* and a non-trivial dependency $\alpha \rightarrow \beta$ causes a violation of BCNF.

  We decompose *R* into:

  - $(\alpha \cup \beta)$
  - $(R - (\beta - \alpha))$

On decomposing a Relation R ,one or more of the resulting schemas may not be in BCNF. In such cases, further decomposition is required, the eventual result of which is a set of BCNF schemas.

- In our example,
- **Inst_dept (ID, Name, Salary, Dept_Name, Building, Budget)**
- **Assume** ***dept_name →building, budget*** *violates*
  - α = *dept_name* β = *building, budget*

    and ***inst_dept*** is replaced by
  - $(\alpha \cup \beta)$ = (*dept_name, building, budget*)
  - (*R* - (β - α)) = (*ID, name, salary, dept_name*)
  - R_1(***dept_name, building, budget***)
  - R_2(***ID, name, salary, dept_name***)

Testing for BCNF

- To check if a non-trivial dependency $\alpha \rightarrow \beta$ causes a violation of BCNF
  1. **compute α^+** (the attribute closure of α), and
  2. **verify that it includes all attributes of *R*,**

     that is α is a **super key of *R***.
     - It suffices to **check only the dependencies in the given set F** for violation of BCNF, rather than check all dependencies in **F^+**.
     - We can show that if none of the dependencies in F causes a violation of BCNF, then none of the dependencies in **F^+** will cause a violation of BCNF,either.
- However, procedure does not work **when a relation is decomposed**.
- That is, **it *does not* suffice** to use *F* when we test a relation R_i , in a decomposition of *R*, for violation of BCNF.

Testing Decomposition for BCNF

when R is decomposed into R_i

- To check if a relation R_i in a decomposition of R is in BCNF,
  - Either test R_i for BCNF with respect to the **restriction** of F to R_i (that is, all FDs in F^+ that contain only attributes from R_i)

OR

  - use the original set of dependencies F that hold on R, but with the following test: (**this method we use in further discussion**)
    - for every set of attributes $\alpha \subseteq R_i$, check that α^+ (the attribute closure of α) either includes **no** attribute of R_i- α, **or** includes **all** attributes of R_i.

Example: $R = (A, B, C)$

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

R is not in BCNF and we got Decomposition $R_1 = (A, B)$, $R_2 = (B, C)$

Is R_1 and R_2 in BCNF ?

Example: $R = (A, B, C)$

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

R is not in BCNF and we got Decomposition

$R_1 = (A, B)$, $R_2 = (B, C)$ **Is R_1 and R_2 in BCNF ?**

Take R_1 all subsets of R_1 are A , B ,AB

1. $A^+ = \{A,B,C\}$ $R_1 - A = B$

A^+ do not include any attribute of **$R_1 - A$ is- FALSE**,

but

A^+ include ALL attributes of R_1 is- TRUE

2. $B^+ = \{B,C\}$ $R_1 - B = A$

B^+ do not include any attribute of **$R_1 - B$ is- TRUE**

R_1 is in BCNF

α^+ either includes **no** attribute of $R_i - \alpha$,
or includes **all** attributes of R_i.

Example: continued..

***R* = (*A, B, C*)**

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

***R* is not in BCNF and we got Decomposition**

R_1 = (*A, B*), R_2 = (*B, C*) **Is R_1 and R_2 in BCNF ?**

Take R_2 all subsets of R_2 are B,C ,BC

1. $B^+ = \{B,C\}$ $R_2 - B = C$

B^+ do not include any attribute of **$R_2 - B$ is- FALSE**,

but

B^+ include ALL attributes of R_2 is- TRUE

2. $C^+ = \{C\}$

C^+ do not include any attribute of **$R_2 - C$ is- TRUE**

R_2 is in BCNF

α either includes no attribute of R_i-

α, or includes all attributes of R_i.

BCNF Decomposition Algorithm

A **general method to decompose** a relation schema so as to satisfy BCNF.

result := { *R* } ;
done := false ;
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 α^+ **does not in** R_i **AND** $\alpha \cap \beta = \varnothing$
 ***result* := (*result* – R_i) $\cup$ (R_i – β) $\cup$ (α, β) ;**
 end
 else *done* := **true ;**

.

The decomposition that this algorithm generates is **not only in BCNF**, but is also a **lossless decomposition**.

Example of BCNF Decomposition

Suppose we have a schema *R* and a non-trivial dependency $\alpha \rightarrow \beta$ causes a violation of BCNF.

According to algorithm, we decompose *R* into:

$$result := (result - R_i) \cup (R_i - \beta) \cup (\alpha , \beta)$$

- ***class* (*course_id, title, dept_name, credits, sec_id, semester, year, building, room_number, capacity, time_slot_id*)**
- Functional dependencies:
  - *course_id*→ (*title, dept_name, credits*)
  - *(building, room_number)*→*capacity*
  - *(course_id, sec_id, semester, year)*→(*building, room_number, time_slot_id)*
- A candidate key {*course_id, sec_id, semester, year*}.

BCNF Decomposition (Cont.)

- BCNF Decomposition:
  - ***course_id→ title, dept_name, credits*** holds
    - but ***course_id*** **is not a superkey**.
  - We replace *class* by:
    - ***Course*** (*course_id*, *title*, *dept_name*, *credits*)
    - ***Class-1*** (*course_id*, *sec_id*, *semester*, *year*,

      building, *room_number*, *capacity*, *time_slot_id)*

BCNF Decomposition (Cont.)

- *course* is in BCNF
  - How do we know this?
- *building*, *room_number*→*capacity* holds on *class-1*
  - but {*building*, *room_number*} is not a superkey for *class-1*.
  - We replace *class-1* by:
    - ***classroom*** **(*building*, *room_number*, *capacity*)**
    - ***section*** **(*course_id*, *sec_id*, *semester*, *year*,**

      building*, room_number*, *time_slot_id*)
- *classroom* and *section* are in BCNF.

Note: It is not always possible to get a BCNF decomposition that is dependency preserving

Boyce- Codd Normal Form

A relation schema ***R*** is in **BCNF** with respect to a set *F* of functional dependencies if for all functional dependencies in F^+ of the form

$$\alpha \rightarrow \beta$$

where $\alpha \subseteq R$ and $\beta \subseteq R$, **at least one of the** following holds:

- $\alpha \rightarrow \beta$ is trivial (i.e., $\beta \subseteq \alpha$)
- α is a super key for *R*

Example schema *not* in BCNF:

instr_dept (*ID, name, salary, dept_name, building, budget*)

Assume F={ *ID* → *name, salary , dept_name, dept_name* → *building, budget* }

instr_dept *is* ***not in BCNF*** because *dept_name*→ (*building, budget)*

BECAUSE, *dept_name*→ (*building, budget)* ***not trivial*** *and* ***dept_name*** **is not a superkey**

Decomposing a Schema into BCNF

- Suppose we have a schema ***R*** and a **non-trivial** dependency $\alpha \rightarrow \beta$ causes a violation of BCNF.

  We decompose *R* into:

  - **$(\alpha \cup \beta)$**
  - **$(R - (\beta - \alpha))$**

On decomposing a Relation R ,one or more of the resulting schemas may not be in BCNF. In such cases, further decomposition is required, the eventual result of which is a set of BCNF schemas.

- In our example,
- **Inst_dept (ID, Name, Salary, Dept_Name, Building, Budget)**
- **Assume *dept_name →building, budget*** *violates*
  - α = *dept_name* β = *building, budget*

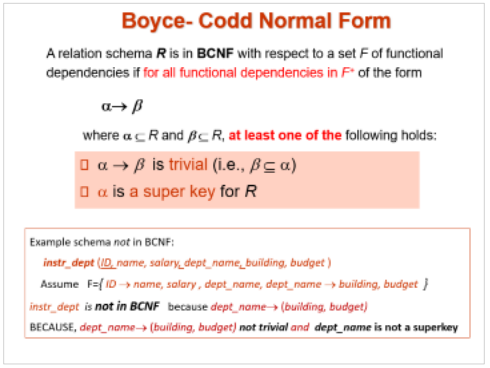
    and ***inst_dept*** is replaced by
  - $(\alpha \cup \beta)$ = (*dept_name, building, budget*)
  - $(R - (\beta - \alpha))$ = (*ID, name, salary, dept_name*)
  - **R_1(*dept_name, building, budget*)**
  - **R_2(*ID, name, salary, dept_name*)**

Testing a Relation for BCNF

- To check if a **non-trivial dependency** $\alpha \rightarrow \beta$ causes a violation of BCNF
  1. **compute** α^+ (the attribute closure of α), and
  2. **verify that it includes all attributes of *R*,**

     that is α is a **super key of *R***.
  - It suffices to **check only the dependencies in the given set F** for violation of BCNF, rather than check all dependencies in $\mathbf{F}^+$.
- However, this procedure does not work **when a relation is decomposed**.
- That is, **it *does not* suffice** to use *F* when we test a relation R_i , in a decomposition of *R*, for checking violation of BCNF.

Example



- $R = (A, B, C)$
  $F = \{A \rightarrow B,\ B \rightarrow C\}$
  Key = $\{A\}$
- R is not in BCNF (why?)
- Assume R is Decomposed into $\boldsymbol{R_1} = (A, B)$, $\boldsymbol{R_2} = (B, C)$
  - $\boldsymbol{R_1}$ and $\boldsymbol{R_2}$ in BCNF (check*) (see notes section)s
  - Lossless-join decomposition (check)
  - Dependency preserving (check)

* Using F it is not possible to check whether R_1 and R_2 are in BCNF. We need functional dependency set F_1,F_2 corresponding to R_1 and R_2 respectively.

Testing Decomposition for BCNF

when R is decomposed into R_i

- To check if a relation R_i in a decomposition of R is in BCNF,
  - Either test $\mathbf{R_i}$ for BCNF with respect to the **restriction** of F to $\mathbf{R_i}$ (that is, all FDs in F^+ that contain only attributes from R_i)

OR

  - use the original set of dependencies F that hold on R, but with the following test: (**this method we use in further discussion**)
    - for every set of attributes $\alpha \subseteq R_i$, check that α^+ (the attribute closure of α) either includes **no** attribute of $(R_i - \alpha)$, **or** includes **all** attributes of R_i.

Example: $R = (A, B, C)$

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

R is not in **BCNF** and we got Decomposition $R_1 = (A, B)$, $R_2 = (B, C)$

Is R_1 and R_2 in BCNF ?

Example: $R = (A, B, C)$

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

R is not in BCNF and we got Decomposition

$R_1 = (A, B)$, $R_2 = (B, C)$ **Is R_1 and R_2 in BCNF ?**

Take R_1 all subsets of R_1 are A , B ,AB

1. $A^+ = \{A,B,C\}$ $R_1 - A = B$

A^+ do not include any attribute of **$R_1 - A$ is- FALSE**

(because $B \in R_1$), but

A^+ include ALL attributes of R_1 is- TRUE

2. $B^+ = \{B,C\}$ $R_1 - B = A$

B^+ do not include any attribute of **$R_1 - B$ is- TRUE**

R_1 is in BCNF

α^+ either includes **no** attribute of $R_i - \alpha$,

or includes **all** attributes of R_i.

Example: continued..

α^+ either includes **no** attribute of R_i- α,
or includes **all** attributes of R_i.

$R = (A, B, C)$

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

R is not in BCNF and we got Decomposition

$R_1 = (A, B)$, $R_2 = (B, C)$ ***Is*** **R_1 and R_2 in BCNF ?**

Take R_2 all subsets of R_2 are B,C ,BC

1. $B^+ = \{B,C\}$ $R_2 - B = C$

B^+ do not include any attribute of **$R_2 - B$ is- FALSE**

(because C$\in$ R_1), but

B^+ include ALL attributes of R_2 is- TRUE

2. $C^+ = \{C\}$

C^+ do not include any attribute of **$R_2 - C$ is- TRUE**

R_2 is in BCNF

Comparison of BCNF and 3NF

- It is always possible to decompose a relation into a set of relations that are in 3NF such that:
  - the decomposition is lossless
  - the dependencies are preserved
- It is always possible to decompose a relation into a set of relations that are in BCNF such that:
  - the decomposition is lossless
  - it may not be possible to preserve dependencies.

END