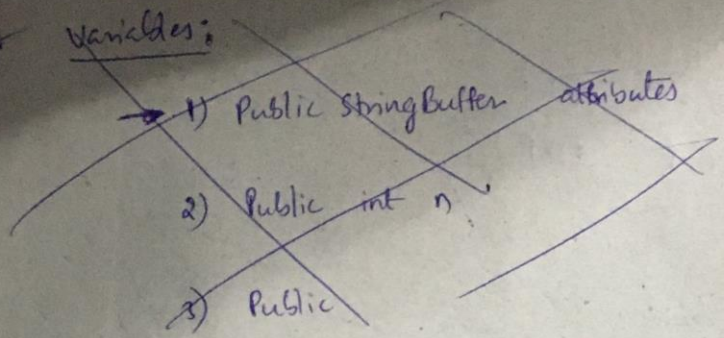
* Variables:



---

* variables

1) Public StringBuffer attributes;

→ ~~Since this~~ i

→ This contains ~~all~~ the attributes of the Relation.

Eg: "ABCDEFG"

Here A, B, C, ... G are all attributes

---

2) Public int n;

↳ Total number of attributes

ie,

n = attributes.length();

---

3) Public String[] lhs;

4) Public String[] rhs;

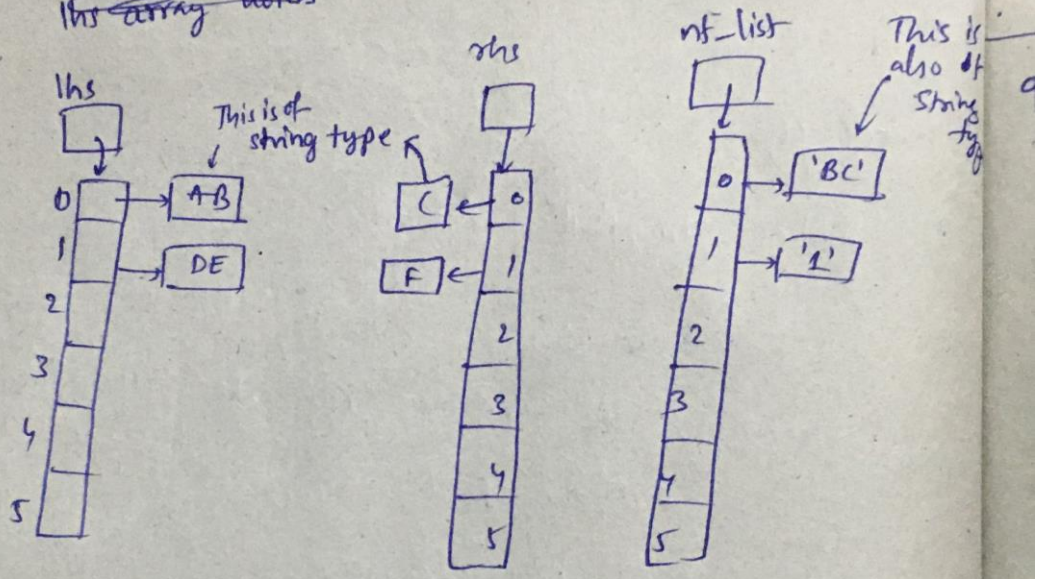
5) Public String[] nf_list;

Eg: consider we have 2 FDS.

AB → C BCNF

DE → F 1NF

Here ~~this string holds~~.



---

6) Public int K=0;

↳ This variable holds the total number of functional dependencies (FD closure ka number)

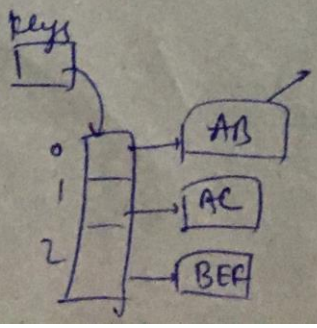
→ Here K means not 'key'.

---

7) Public String[] keys;

↳ This "keys" is of type String[].

→



- These are of type String
- These are all keys

8) Public String keyAttributes

↓

- keyAttributes holds all the key attributes
- Eg: ABCEF (from previous Example ~~of 'F'~~)

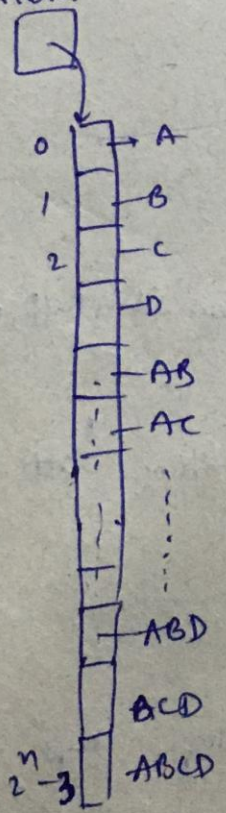
---

9) Public String[] attr_combns.

→ attr_combns contains all the possible ~~to~~ 2^n combinations of attributes.

eg: Attributes = ABCD.

attr_combns



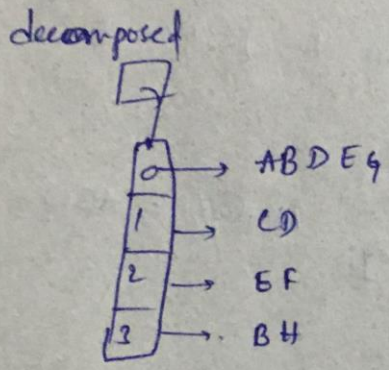
→ This is useful for finding keys.

10) Public Attr[] decomposed.

↳ This is of type Attr[]

→ This contains tables / Relations of ~~*~~ just like parent table.

eg:



i). ABCDEFGH

---

11) Public int r=0.

12) Public ~~int~~ StringBuffer b = new StringBuffer();

These are useful while filling "attr-combns".

---

13) int h=0.

→ useful while filling "keys".

---

14) int l;

↳ number of keys.

15) Private int P = 1;

↳ this holds the total number of decomposed Relations.

---

methods:

i). Private String sort (String s).

→ Takes a string, sorts it & returns the sorted one

eg; i/p = BACDE

o/p = ABCDE.

---

ii) Private String removeDuplicates (String s)

→ i/p = ABCCCDE

o/p = ABCDE

---

iii) Public void removeRedundantFDs (lhs[] . rhs[])

i/p ; → AB → C

→ ABD → C

o/p ; → AB → C

→ null

iv) Public void removeRedundantFDs (lhs, rhs, x)

→ This is overloaded,

→ if (x == 1)

then it clears nf_list also.

---

v) Public void remove nullspaces (lhs, rhs, x)

→ This removes null spaces ~~are~~ that are created in above methods

---

vi) Public void fd closure ( )

→ calculates FD closure

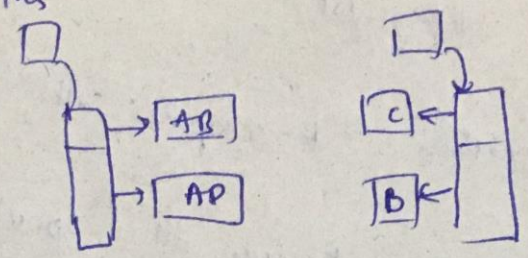
---

vii) Public void fillFDs (→).

→ This takes the FDs & in string format & splits them & stores in 2 different strings.

Eg: i/p : i) "AB → C"
ii) "AD → B"

o/p = lhs



viii) Public void printFDs( )

ix) Public void fillAttrCombns( )

→ This fills the String Array attr_combns.

→ This filled array is useful for finding the keys.

x) store
xi) nchooseP

} above ~~from~~ method uses these 2 methods. for completing the task.

~~xii)~~

xii) Public void fillKeyList()

→ This method computes all the keys & by going through all possible combinations

→ This method uses the following methods.

xiii) ~~get com~~

Private String getCombination()

↳ This method takes ~~a com~~ an appropriate combination from attr-combns and returns it to "fill key list" method.

---

xiv) Public void print keys()

→

xv) Public boolean isAKey()

xvi) Public boolean isAKeyAttribute().

The above 2 methods are useful while finding the Normal form.

---

xvii) Public void fillNormalFormList()

→ attaches a normal form to every FD in FD closure

---

xviii) Public void PrintNF()

NF = Normal form.

---

xix) Public void decomposeRel()

→ In order to increase the NF, this method decomposes the table & stores the ~~doco~~ decomposed tables in Attr[].

---

xx) Public void PrintRel()

→ prints ~~all~~ all the decomposed Relations

---

xxi) Public void CompleteFormalities().

→ i) computes FD closure

→ ii) computes minimal keys

→ iii) computes Normal form of every FD.

by invoking necessary method.

→ if(task.equals("print"))

→ it prints the above '3' things.

xxii) Public boolean calculate FDclosure..... original()

{

calculates FD closure from decomposed rels &
compares it with original FD closure.

}