08. Relational Database Design

[PM Jat, DAIICT, Gandhinagar]

Normal Forms

Normal forms are used as measure of "goodness" of a relation. Higher the normal form, less redundancies, and less anomalies it has; and better the relation is. The process of having "good relations" in a relational database schema is called *Normalization*. Normalized relations have minimized data redundancies and hence minimized anomalies.

Initially Edgar (Ted) F. Codd proposed three normal forms, which he called First, Second, and Third normal forms. A stronger definition of 3NF, called Boyce-Codd Norm Form (BCNF) was proposed later by Boyce and Codd. All these normal forms are based on the functional dependencies among the attributes of a relation. Later Fourth normal form (4NF) and Fifth normal form (5NF) were proposed based on multi-value dependencies and join dependencies respectively.

First Normal Form (1NF):

A relation is in First normal form if it qualifies to following-

- All attributes are "Atomic", that is
    - No composite
    - No Multiple values

With given definition, following relation is not in 1NF-

User(UID, Name, Address, phone)

And suppose it has following two tuples -
<101, Amit, Delhi, {9712312121, 9324121120}>
<102, Mukesh, Mumbai, {9410120220}>

Not in 1NF, because attribute "Phone Number Set" is not atomic.

Alternative, following scheme is also not in 1NF-

User(UID, Name, Address, phone1, phone2)

<101, Amit, Delhi, 9712312121, 9324121120>
<102, Mukesh, Mumbai, 9410120220, null>

Because it just fooling around of previous one. Basically *Phone* is a multi-value attribute. Creating multiple attributes for a multi-value attribute is wrong; by doing so we lose out the fact that these are multiple values of a same attribute.

Suppose we allow having a relation
User(UID, Name, Address, phone1, phone2)

Let us see what is undesirable consequence of such a design? How do we delete a phone number? How do we add a phone number? How do express list the people having no phone number?

With above understanding, following relation schema is also not in 1NF-

Department(DNO, DName, MGRSSN, DLOCATIONS)
[A sample tuple: <4,'Marketing',101, {Delhi,Mumbai,Pune}>]

Composite attributes are also not atomic therefore following relation is also not in 1NF-

Student(ID, Name(Fname, Minit, Lname), Batch, CPI)
[Sample tuple: <200701001,<Charu,K,Chawla>, 2007, 6.8>]

"Contradiction": is not DOB is a composite in following relation scheme?

Student(ID, FName, MInit, LName, DOB, Batch, CPI)
[Sample tuple: <200701001, Charu, K, Chawla, <1988,11,21>, 2007, 6.8>]

is this relation still in 1NF?

It is true that it is composite, but we treat it as single of "Date" type? With the acceptance of user defined object types as attribute type, CJ Date says that an attribute is atomic if it has a single value of "appropriate type".

Modern Object RDBMS, allows having Object Types, Arrays, and even Relation as data types for an attribute. Such attribute type is definitely not atomic in classic sense; however with Object oriented understanding of types, where types take care of manipulating data within object through operator overloading or so, usage of such data type can come nearer to "atomic values".

Again to repeat that Normal Form is a measure of goodness of a relation; each normal form defines its own sets of requirements, and if a relation meets those requirements, then it is in that normal form. It should also be noted that rules are designed such that if a relation is in higher norm form, it is implied to be in lower normal forms.

In this section, we discuss first set of normal forms, i.e. 2NF, 3NF, and BCNF. Requirements for these three normal forms are specified in terms of Functional Dependencies. There are two ways in which these forms are studies in the literature -

1. Begin with BCNF (most stricter form), and come down to 2NF, or
2. Reverse, i.e. start with 2NF, and go up to BCNF

In most modern text, first approach is preferred; we will also follow this route. Probably with following reasons -

- Most relations that you design using your common sense, or by mapping ER to Relations; are likely to be in BCNF..
- Second, 2NF and 3NF are discussed more for historical reasons.

Boyce-Codd Normal Form (BCNF)

A relation R is in Boyce-Codd Normal Form, when determinant of every FD that holds on R, is super-key# of R. In other words, For every FD **A → B** that holds on relation R, A is its super-key key. This requirement is to be checked for all FDs on R, if any FDs fail to meet this criteria, the relation is not in BCNF.

> # In most places in literature; there is usage of super-key; but when we work on minimal set; where we have all left sides irreducible, we can read super-key as Key.

Exercise ##:

Is RXIT(studid, name, progid, cpi, pname, intake, did, dname) in BCNF?

```
studid→ name
studid→ cpi
studid→ progid
progid→ pname
progid→ did
progid→ intake
did  → dname
```

Key: ?

BCNF? <Yes/No>

Exercise ##

Is SP(studid, name, progid, cpi, pname, intake, did) in BCNF?

```
studid→ name
studid→ cpi
studid→ progid
progid→ pname
progid→ did
progid→ intake
```

~~did → dname~~

Key: ?

BCNF? <Yes/No>

Exercise ##:

Is Student(studid, name, progid, cpi) in BCNF?

studid→ name

studid→ cpi

studid→ progid

~~progid→ pname~~

~~progid→ did~~

~~progid→ intake~~

~~did → dname~~

Key: ?

BCNF? <Yes/No>

Exercise ##:

Is PD(progid, pname, intake, did, dname) in BCNF?

~~studid→ name~~

~~studid→ cpi~~

~~studid→ progid~~

progid→ pname

progid→ did

progid→ intake

did → dname

Key: ?

BCNF? <Yes/No>

Exercise ##:

Is Program(progid, pname, intake, did) in BCNF?

~~studid→ name~~

~~studid→ cpi~~

~~studid→ progid~~

progid→ pname

progid→ did

progid→ intake

~~did → dname~~

Key: ?

BCNF? <Yes/No>

Exercise ##:

Is Department(did, dname) in BCNF?

~~studid→ name~~

~~studid→ cpi~~

~~studid→ progid~~

~~progid→ pname~~

~~progid→ did~~

~~progid→ intake~~

did → dname

Key: ?

BCNF? <Yes/No>

Exercise ##: Given R(ABCDEF), and following FDs on R, determine if R is in BCNF?

A→ B

A→ C

A→ D

B → E

B → F

Exercise ##: Given R(ABCDEF), and following FDs on R, determine if R is in BCNF?

A→ B

A→ C

A→ D

AB → E

AB → F

Company Database:

$\text{ssn} \rightarrow \{\text{name, salary, dob, dno, superssn, dname, mgrssn}\}$

$\text{dno} \rightarrow \{\text{dname, mgrssn}\}$

$\text{pno} \rightarrow \{\text{pname, plocation, dno, dname, mgrssn}\}$

$\{\text{ssn, pno}\} \rightarrow \text{hours}$

$\{\text{ssn, dep\_name}\} \rightarrow \{\text{dep\_gender, dep\_bdate, relationship}\}$

Is R(SSN, Name, Salary, DOB, DNO, DName, MGRSSN) in BCNF?

Are Following relations in BCNF?

- Employee(<u>ssn</u>, name, salary, DoB, superssn, dno)
- Department(<u>dno</u>, dname, mgrssn)
- dept_locations(<u>dno, dlocation</u>)
- Project(<u>pno</u>, pname, plocation, dno)
- works_on(<u>essn</u>, <u>pno</u>, hours)
- dependent(<u>essn</u>, <u>dep_name</u>, dep_bdate, relationship)

DA-ACAD Database:

Are following relations in BCNF?

- Student(StudetID, StdName, ProgID, Batch, CPI)
- Term(AcadYear, Semester)
- Course(CourseNo, CourseName, Credit)
- Faculty(FacultyID, FacultyName)
- Offers(AcadYear, Semester, CourseNo, FacultyID)
- Registers(StudetID, AcadYear, Semester, CourseNo, course_grade)
- Result(StudetID, AcadYear, Semester, Semester_SPI, Semester_CPI)

$\text{StudetID} \rightarrow \{\text{StdName, ProgID, Batch, CPI}\}$

$\text{CourseNo} \rightarrow \{\text{CourseName, Credit}\}$

$\text{FacultyID} \rightarrow \text{FacultyName}$

$\{\text{StudetID,AcadYear,Semester}\} \rightarrow \text{Semester\_SPI}$

$\{\text{StudetID,AcadYear,Semester}\} \rightarrow \text{Semester\_CPI}$

$\{\text{StudetID,CourseNo,AcadYear,Semester}\} \rightarrow \text{course\_grade}$

$\{\text{CourseNo,AcadYear,Semester}\} \rightarrow \text{FacultyID}$

TGMC database

- Member(MembID, MembName, MembEmail, TeamID)
- Team(TeamID, TeamPWD, MentorID)
- Mentor(MentorID, MentorName, MentorEmail, InstID)
- Institute(InstID, InstName, City, PIN, State)

All Attributes:

MembID , MembName, MembEmail, TeamID, TeamPWD, MentorID, MentorName, MentorEmail, InstID, InstName, City, PIN, State

Given FDs:

- MembID →{MembName, MembEmail, TeamID, TeamPWD, MentorID, MentorName, MentorEmail, InstID, InstName, City, PIN, State}
- TeamID → {TeamPWD, MentorID, MentorName, MentorEmail, InstID, InstName, City, PIN, State}
- MentorID → {MentorName, MentorEmail, InstID}
- InstID → {InstName, City, PIN, State}
- PIN → {City, State}

3rd Normal Form (3NF)

Third Normal Form is less restrictive than BCNF; it relaxes BCNF condition for *prime attributes* (attribute that are part of some candidate key).

A relation is in 3NF, if, for every FD **A** → **B** that holds on relation R,

- A is its super-key, or
- B is a prime attribute.

An attribute is prime if it is part of some [candidate] key.

Exercise #10: A relation R(A,B,C} having following FDs: {A,B} → C, C→A

> {A, B} is key.
>
> Relation is not in BCNF but in 3NF?

Note that 3NF definition is relaxed with respect to BCNF. Therefore if a relation is BCNF, it is automatically in 3NF?

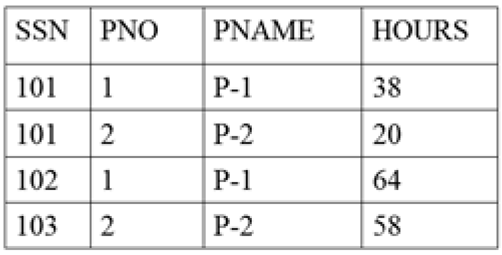
Exercise #12: Suppose we have WORKS_ON as following:

WORKS_ON(ESSN, PNo, PName, Hours)

FDs (suppose):

- PNO → Pname
- PName → PNo
- {ESSN, PNo} → Hours
- {ESSN, PName} → Hours

Keys: ?



SSN	PNO	PNAME	HOURS
101	1	P-1	38
101	2	P-2	20
102	1	P-1	64
103	2	P-2	58

Is it in BCNF?

Is it in 3NF?

> Key: Two keys. {ESSN, PNO}, and {ESSN, PName}
>
> Is it in BCNF? No. First two FDs violate the requirement.
>
> Is it in 3NF? Yes. Right Attributes are Prime Attributes.

Exercises #12: find out if following relations are in 3NF?

Given relation R(SSN, FName, PNO, PName, HOURS), and FD set-

{SSN, PNO} → HOURS

SSN → FNAME

PNO → PNAME

SSN	FNAME	PNO	PNAME	HOURS
101	Sumit	1	P-1	38
101	Sumit	2	P-2	20
102	Vipul	1	P-1	64
103	Ajay	2	P-2	58

Compute key?

Is R in BCNF?

Is R in 3NF?

> Key: {SSN, PNO}
>
> BCNF? No. Last two FDs violate the requirement!
>
> 3NF? No. Again last two FDs violate the requirement!

Exercise #13: is our SP (student-program combined) in BCNF? is it in 3NF?

Second Normal Form (2NF)

It is classic definition and defined in terms of Key. Second Normal form requires that all non-prime attributes are irreducibly (fully) dependent on key; that is no non-prime attribute should be partially dependent on key.

> Note: These dependencies are also checked using FDs.

With this definition does our SP in 2NF?

Yes. It is.

Below is example that is not in 2NF?

AB → C, A → D; Key is AB. C and D are non-prime attributes. C is dependent on Key, while D is partially dependent on Key, as there is FD A → D

Note that this definition of 2NF is relaxed with respect to 3NF. Therefore if a relation is 3NF, it is automatically in 2NF

Exercise #14: Does R of our exercise #12 in 2NF?

Exercise #15: A → B, A → C, C → D; is this in 2NF?

Yes.

Key is A; and non-prime attributes are B, C, and D.

B is dependent on Key; FD $A \rightarrow B$.

C is dependent on Key; FD $A \rightarrow C$.

D is dependent on Key; $A \rightarrow D$. (transitively inferred from $A \rightarrow C$ and $C \rightarrow D$

Alternate Definition of 3NF from second normal form side

3NF definition: all non-prime attributes are irreducibly and directly dependent on key, transitively dependency is not allowed.

Note that both 3NF(i.e. earlier and this) mean same?

Does every relation that is in 3NF, also in 2NF?

Yes.

Had we have an $A \rightarrow D$ instead of $C \rightarrow D$ exercise #14, and then it would have been in 3NF, and BCNF as well. In fact 3NF require every non-prime attributes to be dependent on Key (and every Key) and directly – no transitively.

BCNF also essentially requires that every attribute including prime attributes should be dependent on every key.

Exercise #16: Compute Normal Form for
R(CourseNo, Sem, AcadYear, InstructorID, StudentID, Grade)

Some Normal Form theorems

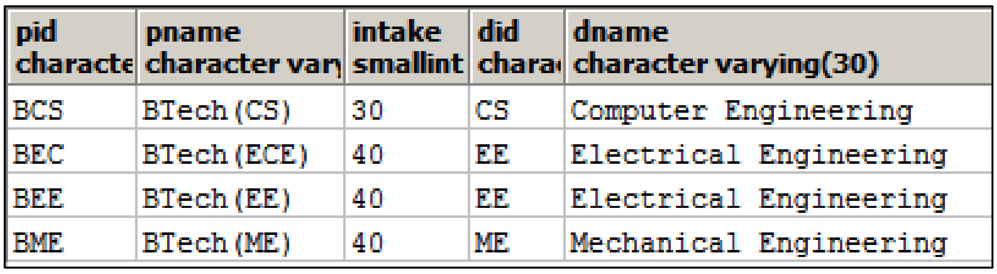
- All attribute Key Relation is always in BCNF
    - If a relation has no FD than key is all attributes, and such relations are always in BCNF
- Also, A relations with only two attribute is always in BCNF

Determine Normal Form of a Relation (the method)

```
Input: Relation R, set of FDs F
Output: Highest Normal Form of a relation
Method:
    Compute all keys from F
    //Check for BCNF
    If there is any* FD f that violates BCNF requirement THEN
        State "relation is not in BCNF, FD f violates"
        Also output other FDs that are violating BCNF requirement.
        //Check for 3NF
        If there is any* FD that violates 3NF requirement THEN
            State "relation is not in 3NF, FD f violates"
            Also output other FDs that are violating 3NF requirement.
            //Check for 2NF
            If there is any* FD that violates 2NF requirement THEN
```

Decomposition

How do you make the relations in higher normal form?
Our following relation PD is in 2 NF; how do you make it in 3NF?



pid characte	pname character vary	intake smallint	did chara	dname character varying(30)
BCS	BTech(CS)	30	CS	Computer Engineering
BEC	BTech(ECE)	40	EE	Electrical Engineering
BEE	BTech(EE)	40	EE	Electrical Engineering
BME	BTech(ME)	40	ME	Mechanical Engineering

Probably, you already know the answer? P(PID,PName,Intake,DID) and D(DID,DName)!
Note the repetition of DID in both relations?

How do we decide the split for following relations?

R(ABCD), {AB → C, A → D}

R(ABCD), {A → B, B → C, C → D }

Requires us to have some systematic methodology for split; the process of splitting a relation into multiple relations is called "decomposition".

Bottom-up relation Design Approach:

We begin with a universal relation that is a relation with all attributes of database. Apply decomposition algorithms; compute multiple relations that are in desirable normal form.

Firs set of decomposition algorithms are based on Functional dependencies. We will see them here.

Decomposition Requirements

Decomposition of a relation R into R1, R2, R3, … Rm implies, attributes of R spread in said m relation schemas. Any arbitrary decomposition does not serve any purpose rather adds noice to the data. A decomposition needs to comply following requirements-

- Loss-less: JOIN of decomposed relation gives back original relation; suppose r1, r2, r3, …. rm are instances of decomposed relation schemas, then r1 * r2 * r3 …. * rm should yield r of R. [discussed below]
- Attribute Preserving: Union of attributes of all decomposed relation should be equal to attributes of R
- FD Preserving: A FD is said to be lost if its attributes are split into multiple relations. It is desirable that we should not be losing any FD from minimal set; it is ok loosing inferred FD.

Lossless Decomposition:

As already stated, decomposition should guarantee r = r1 * r2 . When it does, then it is "loss-less join decomposition" or simply "loss-less decomposition".

Let us say R(A1,A2,A3,A4,A5,A6) getting decomposed into R1(A1,A2,A3,A4) and R2(A4,A5,A6); that if r is relation instance of R, then instances of R1 and R2 will be computed as following-

$$\mathbf{r1 \leftarrow \pi_{a1,a2,a3,a4}(r), \text{ and}}$$

$$\mathbf{r2 \leftarrow \pi_{a4,a5,a6}(r)}$$

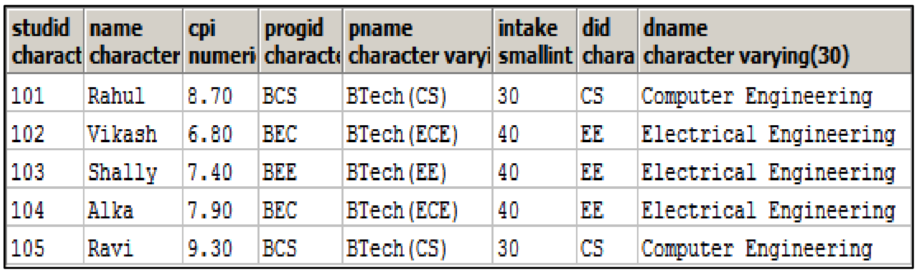
And lossless decomposition should guarantee **r = r1 * r2**

Consider SPD example, and see if they meet out on above requirements?

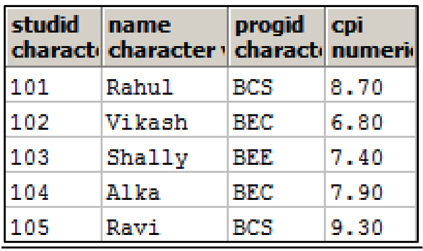
See if

- Lossless : spd = s * p * d
- Attribute Preserving : SPD = S ∪ P ∪ D
- FD preserving: F = Fs ∪ Fp ∪ Fd

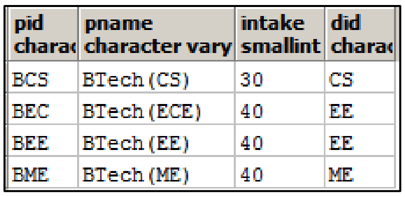
```
studid → name

studid →
prog_id

studid → cpi

studid → pid

pid → pname

pid → intake

pid → did

did → dname
```



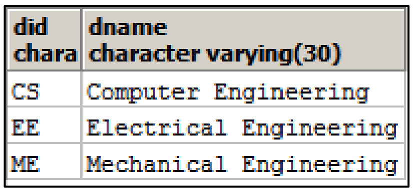
studid charact	name character	cpi numeri	progid characte	pname character varyi	intake smallint	did chara	dname character varying(30)
101	Rahul	8.70	BCS	BTech(CS)	30	CS	Computer Engineering
102	Vikash	6.80	BEC	BTech(ECE)	40	EE	Electrical Engineering
103	Shally	7.40	BEE	BTech(EE)	40	EE	Electrical Engineering
104	Alka	7.90	BEC	BTech(ECE)	40	EE	Electrical Engineering
105	Ravi	9.30	BCS	BTech(CS)	30	CS	Computer Engineering



studid	name	progid	cpi
characte	character v	characte	numeri
101	Rahul	BCS	8.70
102	Vikash	BEC	6.80
103	Shally	BEE	7.40
104	Alka	BEC	7.90
105	Ravi	BCS	9.30



pid	pname	intake	did
charac	character vary	smallint	charac
BCS	BTech(CS)	30	CS
BEC	BTech(ECE)	40	EE
BEE	BTech(EE)	40	EE
BME	BTech(ME)	40	ME



did	dname
chara	character varying(30)
CS	Computer Engineering
EE	Electrical Engineering
ME	Mechanical Engineering

studid → name

studid → prog_id

studid → cpi

studid → pid

pid → pname

pid → intake

pid → did

did → dname

It is called "loss-less" because natural join of decomposed relation results into original relation, that is, there is no loss of "information".

Lossy decomposition normally results into additional tuples (also referred as spurious tuples) on having natural join of decomposed relations.

Example A lossy decomposition

R(PNO, PNAME, ESSN, HOURS)

PNO → PNAME

PNAME → PNO

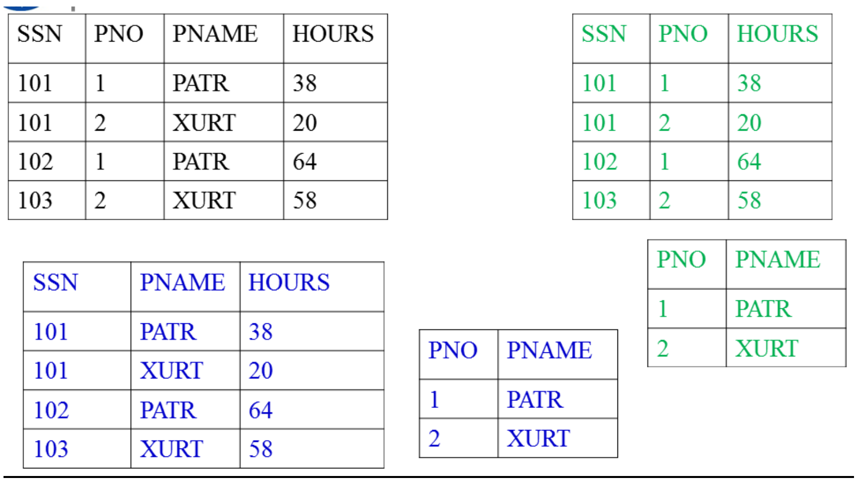
{PNO, ESSN} → HOURS

{PNAME, ESSN} → HOURS

Keys:

{PNO, ESSN},

{PNAME, ESSN}



SSN	PNO	PNAME	HOURS
101	1	PATR	38
101	2	XURT	20
102	1	PATR	64
103	2	XURT	58

SSN	PNO	HOURS
101	1	38
101	2	20
102	1	64
103	2	58

PNO	PNAME
1	PATR
2	XURT

SSN	PNAME	HOURS
101	PATR	38
101	XURT	20
102	PATR	64
103	XURT	58

PNO	PNAME
1	PATR
2	XURT

PNO → PNAME

~~PNAME → PNO~~

{PNO, ESSN} → HOURS

~~{PNAME, ESSN} → HOURS~~

Keys:

{PNO, ESSN},

~~{PNAME, ESSN}~~

SSN	PNO	PNAME	HOURS
101	1	PATR	38
101	2	XURT	20
102	1	PATR	64
103	3	XURT	58

SSN	PNAME	HOURS
101	PATR	38
101	XURT	20
102	PATR	64
103	XURT	58

PNO	PNAME
1	PATR
2	XURT
3	XURT

ON JOIN we do not get back original relation

SSN	PNO	HOURS
101	1	38
101	2	20
102	1	64
103	3	58

PNO	PNAME
1	PATR
2	XURT
3	XURT

A test for binary decomposition: Let us say R is decomposed into if R1 and R2, then it is lossless, when we have one of following FD-

- (R1 intersection R2) → (R1-R2), or
- (R1 intersection R2) → (R2-R1), or

Or in other words, common attribute(s) are key of one of the decomposed relation.

Decomposition Algorithms

- BCNF decomposition algorithm
- 3NF synthesis algorithm

BCNF decomposition algorithm

- //Source: Ullman et al
- Input: A relation **R** with set (minimal) of FD's **F**.
- Output: A decomposition of R into a collection of relations, all of which are in BCNF.

```
NR ← {(R,F)}
Repeat till each relation S in NR is in BCNF.
    If there is a FD X → Y over relation S that violates BCNF condition.
        Compute X+, and choose X+ as one relation as S1, and another S2 as {(S - X+) U X}
        Map FD set Fs on S1 and S2 and compute Fs1 and Fs2
        Replace <S, Fs> in NR with <S1,Fs1> and <S2,Fs2>
Recursively repeat the algorithm on S1 and S2
```

Computation of Projected FDs

Suppose you have a relation R and FD set F on R; let us say R is split into R1 and R2, FDs on R1 and R2 also needs to be projected. Done as following-

- For every FD X → Y on R, of X U R is subset of R1 then X → Y is projected on decomposed relation R1
- This is repeated for every FD in F for every decomposed relation
- At the end we have sets of projected FDs on every decomposed relation.

Consider relation SP(StudID, Name, CPI, ProgID, PName, Intake, DID) and FD set F (given in figure below)

If we break relation SP into S(StudID, Name, CPI, ProgID) and P(ProgID, PName, Intake, DID); based on above understanding we decompose F into Fs and Fp as following. You may note that we could not project some of FDs on either of relation; those FDs are said to be lost. However not really lost, because these are inferred FDs and if we ensure base FDs these are automatically ensured.

F	Fs	Fp
studid → name	studid → name	progid → pname
studid → progid	studid → progid	progid → intake
studid → cpi	studid → cpi	progid → did
studid → progid	studid → progid	
studid → pname		
studid → intake		
studid → did		
progid → pname		
progid → intake		
progid → did		

Exercise #17: Apply BCNF Decomposition Algo

1. Given relation R(SSN, FName, PNO, PName, HOURS), and FD set-
   {SSN, PNO} → HOURS --FD1
   SSN → FNAME --FD2
   PNO → PNAME --FD3

   Key: {SSN, PNO}

   FD violates SSN → FNAME violates BCNF requirement

   Compute SSN^+ = {SSN, FName}

   Have R1(SSN, FName) and R2(SSN, PNO, PName, HOURS), and projected FDs are F1={SSN → FNAME} and F2 = {{SSN, PNO} → HOURS , PNO → PNAME }

   We can prove that R1 is now in BCNF. But R2 is not; therefore we further apply the algorithm on this, FD that violates the requirement is PNO → PNAME; compute closure of PNO, we get {PNO, PName), so we decompose R2 into

   R21(PNO, PName) and R22(SSN, PNO, HOURS)

   Projected FDs are F21 = {PNO → PNAME } and F22 = {{SSN, PNO} → HOURS }

   And, can prove that now R21 and R22 both are in BCNF?

Exercise #18: Decompose following relation using BCNF decomposition Algorithm

R(ABCDE), {AB → C, A → D, A → E }
R1(ADE) and R2(ABC); and FDs are F1={A → D, A → E} and F2={AB → C}
And both are in BCNF!

Decompose following relation using BCNF decomposition Algorithm-

1. R(ABCD), {A → B, B → C, C → D }
2. **Medicine**(TradeName, GenericName, BatchNo, Stock, MRP, TaxRate, Manufacturer)
   TradeName → GenericName
   TradeName → Manufacturer
   BatchNo → TradeName
   BatchNo → Stock
   BatchNo → MRP
   GenericName → TaxRate

3. Book(ISBN, Title, Author, Publisher, Price, AccessonNo);
   AccessonNo → ISBN
   ISBN → {Title, Publisher, Price}

BCNF decomposition algorithm guarantees lossless decomposition but does not guarantee FD preservation. For example, consider relation

R(A,B,C), and FDs
AB→C
C→B

Key: AB, AC

FD C → B violates BCNF requirement, and we have following decomposition

C^+=BC; R1(BC); R2(AC); F1{C→B}; F2{}.

Though both relations are in BCNF, we lose FD AB → C

Other example can be our WH(PNO, PNAME, ESSN, HOURS) with FDs

PNO → PNAME
PNAME → PNO
{PNO, ESSN} → HOURS
{PNAME, ESSN} → HOURS

Note that there can be two BCNF decompositions based on which FD we begin with, and also note that we will be losing one of the FD3 or FD4. Of course both relations will be lossless and leading to relations in BCNF!

3NF Synthesis Algorithm

3-NF Synthesis algorithm guarantees lossless and preserves FDs, however its resultant relations "may not" (but quiet likely to) be in BCNF, but surely in 3NF.

Here is how it goes-

- Find a minimal FD set on R.
- For each set of FDs where X is determinant; say X→A1, X→A2, … X→An; create a relation schema R'(X U A1 U A2 U … U An).
- If none of the relation created in above step that contains key of R; then create one more relation schema for key of R.

Example #19: Given relation R(ABCDEF), and FDs, and using 3NF synthesis algo, we get following decomposition

A → B
A → C
C → D
C → E
	Key: AF	R1(ABC)
		R2(CDE)
		R(AF)

In 3NF synthesis algorithm, sometimes we may get a relation that is subset of another. In such a case subset relation can be dropped. For example, we have a relation

Example #20: Given relation R(A,B,C,D,E), and FDs, and using 3NF synthesis algo, we have following decomposition

{A,B} → C
C → B
A → D
	Key: ABE	R1(ABC)
		~~R2(CB)~~
		R3(AD)
		R4(ABE)

R2 is subset of R1, and can be dropped.

Which relation of this decomposition is not in BCNF?

==> Attempt decomposing it using BCNF decomposition algorithm?

Exercises: Decompose given following relations decompose them using 3NF synthesis algorithms.

1. R(ABCDEF), and FDs
   A → B
   B → CDE
   E → F

2. Book(ISBN,Title,Author,Publisher,Price, AccessonNo), and FDs
   AccessonNo → ISBN
   ISBN → {Title, Publisher, Price}

3. R(CourseNo, Sem, AcadYear, InstructorID, StudentID, Grade)
   {CourseNo, Sem, AcadYear} → InstructorID
   {CourseNo, Sem, AcadYear, StudentID} → Grade

4. LibMember(ID, Name,Type, NoOfBooksCanBeIssued,IssueDuration)

5. R (StudentID, SPI, CPI_UptoDate, CPI_UptoASem, AcadYr, Sem, ProgCode, CourseNo, Grade)

6. IssueLog(IssueDate, MemberID, AccessonNo, DueDate, ReturnDate)