05. 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 it has, and less the anomalies it has; and better the relation is. The process of having “good relations” in a relational database schema, i.e. they are free from anomalies and redundancies is called as Normalization.

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 a 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-

- A relation is set of tuples, i.e. every tuple is distinct
- Each value in a tuple is “atomic” every value is atomic implies that there is NO multivalued attribute

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

Room_HOR(RoomNo, Wing, Floor, Resident_Set)

And suppose it has following two tuples-
<C115, C, 1, {201001023, 201001111}>
<H211, H, 2, {201111011}>

Not in 1NF, because attribute “Resident_Set” is not atomic.

Alternative, following scheme is also not in 1NF-

Room_HOR(RoomNo, Wing, Floor, Resident1, Resident2),

and suppose it has following two tuples -
<C115, C, 1, 201001023, 201001111>
<H211, H, 2, 201111011, null>

Because it just fooling around of previous one. Basically a multi-value attribute, and creating multiple attributes for a multi0value attribute, is in-fact wee loose out the fact that these are multiple values of same attribute.

> Suppose we allow having a relation
> Room_HOR(RoomNo, Wing, Floor, Resident1, Resident2)
>
> Let us see what is undesirable consequence of such a design? Attempt answering following queries: Give us wing wise count of residents?

With this 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 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 requirement than 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 automatically implies that it is in its lower normal forms.

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

1. Begin with BCNF (more stricter form), and come down to 2NF, or
2. Reverse, i.e. start with 2NF, and go upto 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 all left sides are irreducible, we can read super-key as Key.

Exercise #9: figure out if following relations are in BCNF?

XIT database:

- Student(ID,Name, dob, cpi, prog_id)
- Program(pid, pname, intake, did)
- Department(did, dname)

Company Database:

- Employee(<u>ssn</u>, name, salary, DoB, superssn, dno)
- Department(<u>dno</u>, dname, mgrssn, mgrstartdate)
- 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)

TGMC database

- Member(<u>MembID</u>, MembName, MembEmail, TeamID)
- Team(<u>TeamID</u>, TeamPWD, MentorID)
- Mentor(<u>MentorID</u>, MentorName, Email, InstID)
- Institute(<u>InstID</u>, InstName, City, PIN, State)

All Attributes:
MembID , MembName, MembEmail, TeamID, TeamPWD, MentorID, MentorName, Email, InstID, InstName, City, PIN, State

Given FDs:

- TeamID → {TeamPWD, MentorID, MentorName, Email, InstID, InstName, City, PIN, State}
- MentorID → {MentorName, Email, InstID}
- InstID → {InstName, City, PIN, State}
- PIN → {City, State}

3rd Normal Form (3NF)

3NF 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.

NO TRANSITIVE DEPENDENCY

IF A IMPLIES B AND B IMPLIES C THEN A IMPLIES C. THIS IS TRANSITIVE DEPENDENCY AND IT VIOLETS 3NF

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?

Exercise #11: 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

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

Keys: ?

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?

2nd Normal Form (2NF)

It 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.

NO PARTIAL DEPENDENCY

> Note: These dependencies are also checked using FDs.

PARTIAL DEPENDENCY: IF PROPER SUBSET OF CANDIDATE KEY DETERMINES NON-PRIME ATTRIBUTE

With this definition does our SP in 2NF?

Yes. It is.

Below is example that is not in 2NF?

$AB \rightarrow C$, $A \rightarrow D$; 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 $A \rightarrow D$

<u>Exercise #14</u>: $A \rightarrow B$, $A \rightarrow C$, $C \rightarrow D$; 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 $C \rightarrow D$

Exercise #15: Does R of our exercise #12 in 2NF?

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)

<u>Some Normal Form theorems</u>

- 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
                State "relation is not in 2NF, FD f violates"
                Also output other FDs that are violating 2NF requirement.
            Else
                Relation is in 2NF
        Else
            Relation is in 3NF
    Else
        Relation is in BCNF
*Needs to be checked for every FD
```

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-

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

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

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 characte	name character	progid characte	cpi 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

studid → name

studid → prog_id

studid → cpi

studid → pid

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

pid → pname

pid → intake

pid → did

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

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

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

PNO	PNAME
1	PATR
2	XURT
3	XURT

PNO	PNAME
1	PATR
2	XURT
3	XURT

ON JOIN we do not get back original relation

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 $\mathbf{X}^+$, and choose X^+ as one relation as S1, and another S2 as $\mathbf{\{(S - X^+) \cup X\}}$

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

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

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

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

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

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

Multi Value Dependencies or MVDs

Consider relation R(UUID, email, phone). A person can have multiple emails and multiple phone numbers.

What is the key?

In what normal form the relation is?

UUID	email	Phone
101	101@gm.com	91011
101	101@ym.com	91015

Relation is in BCNF, still has some anomalies?

- How do you delete a phone number?
  - Do we set null for the attribute in corresponding tuple? We cannot because the attribute is part of key!
- How do you add a new number?
  - Do we update any of existing tuple (if some value null)?
- The problem is partially solved by making attributes independent, and have following tuples for recording same facts-

UUID	email	Phone
101	101@gm.com	91011
101	101@ym.com	91011
101	101@gm.com	91015
101	101@ym.com	91015

Now to delete a phone number –
"DELETE FROM CITIZEN WHERE UUID = 101 AND Phone = '91011'"

For adding a phone number, we add phone number tuple for all emails of the user, and updated relation becomes as following-

UUID	email	Phone
101	101@gm.com	91011
101	101@ym.com	91011

101	101@gm.com	91015
101	101@ym.com	91015
101	101@gm.com	92015
101	101@ym.com	92015

Obviously you see redundancies and anomalies here?
Motivates us for higher normal form!

Basically the problem is due to a phenomenon called multi-value dependencies? Here you have

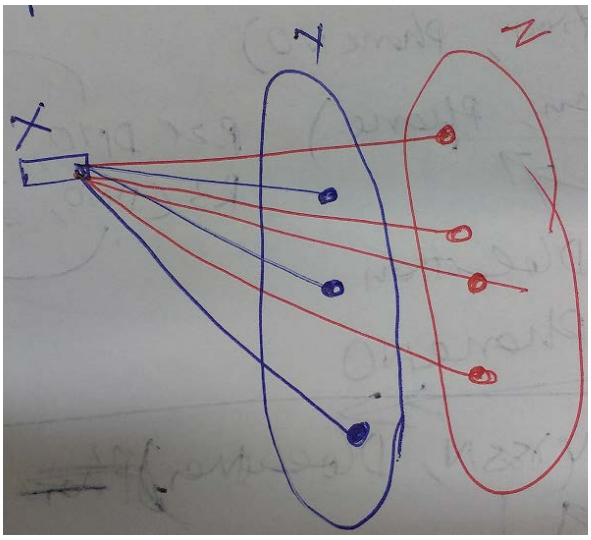
A person (given UUID), has multiple phone numbers?

A person (given UUID), has multiple emails?

You should be able to correlate this with functional dependencies; if for a given value of attribute X, if we have a single value of Y retrieved from database then we say it is FD “X → Y”; if we get a set of values of Y, then it is Multi-Value Dependency or MVD and represented as

X -->> Y

and read as “**X multi-determines Y**”



In our example here, we have following MVDs

UUID -->> email
UUID -->> phone

Note that FD is a special case of MVD, when set of values of Y, for a given value of X, is singleton then it is FD.

Inference rules that we learned for FDs may not be true for MVDs.

We do not plan to look into inference rules for MVDs.

Trivial MVD

- A MVD A -->> B is called trivial, when either of following is true -
  A U B = R, or
  B is subset of A

4NF based on MVDs

- A relation is in 4NF if and only if, when for every non-trivial FD or MVD, key is the determinant.
- In other words, relation is in 4NF, only if it is in BCNF and all MVDs are in fact “FDs out of keys”. [Note: Key is defined only in terms of FDs]
- Definition of 4NF definition covers BCNF.

4NF Decomposition Algorithm

BCNF decomposition algorithm can be extended for BCNF

Input: Relation R and FD set F

NR ← {(R,F)}

Repeat till each relation S in NR is in 4NF.

  If there is a FD X → Y or X -->> Y over relation S that violates 4NF requirement.

    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>, <S2,Fs2>

Recursively repeat the algorithm on S1 and S2

In our example, the relation R(UID, email, phone) with following MVDs: UID -->> email and UID -->> phone; is not in 4NF because

- MVDs are not trivial,
- MVDs are not FDs and UID is not the key

Using above Algorithm can be decomposed as following-

UID	email
101	101@gm.com
101	101@ym.com

`UID -->> email`

UID	Phone
101	91011
101	91015

`UID -->> Phone`

==> Now MVDs are trivial in their respective relations.

Note following-

- MVD should not be a problem in real cases, as most likely be get addressed by other FDs.
- Consider example DNO, DNAME, DLocation, MGRSSN; you have following FDs and MVDs here-

    DNO → {DNAME, MGRSSN}
    DNO -->> DLocation
- Being ignorant about MVD, if we decompose the relation using BCNF decomposition algorithm, what we have is following-
  R1(<u>DNO</u>, DNAME, MGRSSN); FDs: {DNO → DNAME, DNO → MGRSSN}
  R2(<u>DNO, DLocation</u>}; FDs: None; MVD: DNO -->> DLocation
    - Now we look at MVD; projected on R2; we find that it is trivial and has already been taken care of.
    - Both the relations are in 4NF.
- Consider a relation R(SSN, PNO, HOUR) do you see MVDs
  SSN -->> {PNO, HOUR}?

  Is the relation in BCNF? Yes. We have only FD {SSN, PNO} → {HOUR}.

  Is the relation in 4NF? Yes; the MVD is trivial.

- Now let us consider a situation, where we have more than one MVD in a relation. Consider R(SSN, PNO, HOUR, DEP_NAME, DEP_RELATION)

  MVDs
    SSN -->> {PNO, HOUR}
    SSN -->> {DEP_NAME, DEP_RELATION}

  FDs
    {SSN, PNO} → {HOUR}
    {SSN, DEP_NAME} → DEP_RELATION

  Key: {SSN, PNO, DEP_NAME}

  Is it in BCNF? <u>NO</u>

  If we decompose it using BCNF decomposition algo, we get following relations
    R1(SSN, PNO, HOUR)
    R2(SSN, DEP_NAME, DEP_RELATION)

  R1 and R2 are in 4NF as well?
  MVDs are trivial in their respective relations.

==> <u>Do not consider MVDs that are reverse consequence of a FD</u>; for example DNO -->> SSN? you may find many such MVDs otherwise.

5th and higher normal forms

Theoretically, yes there are higher normal forms. Higher normal forms are based on "join dependencies".

If a relation R can be losslessly decomposed into R1, R2 (or more) then R is said to have a join dependency. It is expressed as

*{R1, R2}

It is also said that you can have lossless decomposition of a relation into R1 and R2 only when you have join dependency *{R1,R2}.

Join dependency are hard to detect otherwise but can be expressed in terms of FDs and MVDs.

If a relation R(ABCDE) has FDs A → BC and C → DE then the relation has a join dependency, and is expressed as *{ABC, CDE}.

> You can decompose (lossless) relation EMP_PROJ(SSN, ENAME, PNO, HOURS) into EMP(SSN,ENAME) and PROJ(SSN,PNO,HOURS), because it has a join dependency *{EMP(SSN,ENAME), PROJ(SSN,PNO,HOURS)}

If a relation R(ABCD) has MVDs A -->> {BC} then the relation has a join dependency, and is expressed as *(ABC, AD).

You also have a join dependency when you have FDs A → B and A → C as *(AB, AC) but such join dependency is called as trivial join dependency. JDs implied by candidate key, that means each of decomposed relation is super-key of original relation

A relation is in 5th Normal form if there is no non-trivial join dependency.

Below is a peculiar example (from CJ Date's book) having a join dependency that is not visible through FD and MVD?

Suppose we have following three facts drawn from tuples in relation SPJ

(1) Supplier s1 supplies part p1
(2) Project j1 uses part p1
(3) Supplier s1 supplies some part to j1

SPJ

S#	P#	J#
S1	P1	J2
S1	P2	J1
S2	P1	J1
S1	P1	J1

From these three observations if we can infer that s1 supplies p1 to project j1, then an additional tuple should also exist in relation SPJ? Is not it strange derivation?

SP

S#	P#
S1	P1
S1	P2
S2	P1

PJ

P#	J#
P1	J2
P2	J1
P1	J1

SJ

S#	J#
S1	J2
S1	J1
S2	J1

If so, then there is an insertion anomaly in the relation SPJ. It can be shown that relation SPJ with given constraint, has a 3-way join dependency. It should be easy to establish that drawn above three fact from relations SP, PJ, and SJ respectively, and JOIN of SP, PJ and SJ will give a tuple that expresses that s1 supplies p1 to project j1.