Multi Value Dependencies, 4NF, and Beyond

Source/References:
Database Systems: The Complete Book, and Elmasir/Navathe

pm jat @ daiict

Multi Value Dependencies (MVDs)

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?

Multi-value Dependencies or MVDs

- R(UUID, email, phone) and NO FD
- The relation is in BCNF,
- …. still has redundancies and anomalies?

MVDs causing anomalies

- Suppose person UUID 101, has two emails and two phone. Having following two tuples is not enough for this. Why?

How do you delete one email or phone for UUID=101?

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

MVDs causing anomalies

- Suppose person UUID 101, has two emails and two phones. You need to have following four tuples

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

Now you can delete as following –
DELETE FROM … WHERE UUID=101 and phone=91011 !

MVDs causing anomalies

- Suppose person UUID 101, has two emails and two phone. You nee to have following four tuples. Now what if you need to add one more phone number to 101?

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

You need to repeat new phone number for every email!

Multi-value Dependencies or MVDs

- What is the problem? Relation is BCNF and has still has anomalies!
- The problem is multi-value attributes, and problem when there are more than one Multi-value attributes.
- Formally this kind of dependencies is called as “Multi-Value” dependencies.

Multi Value Dependencies or MVDs

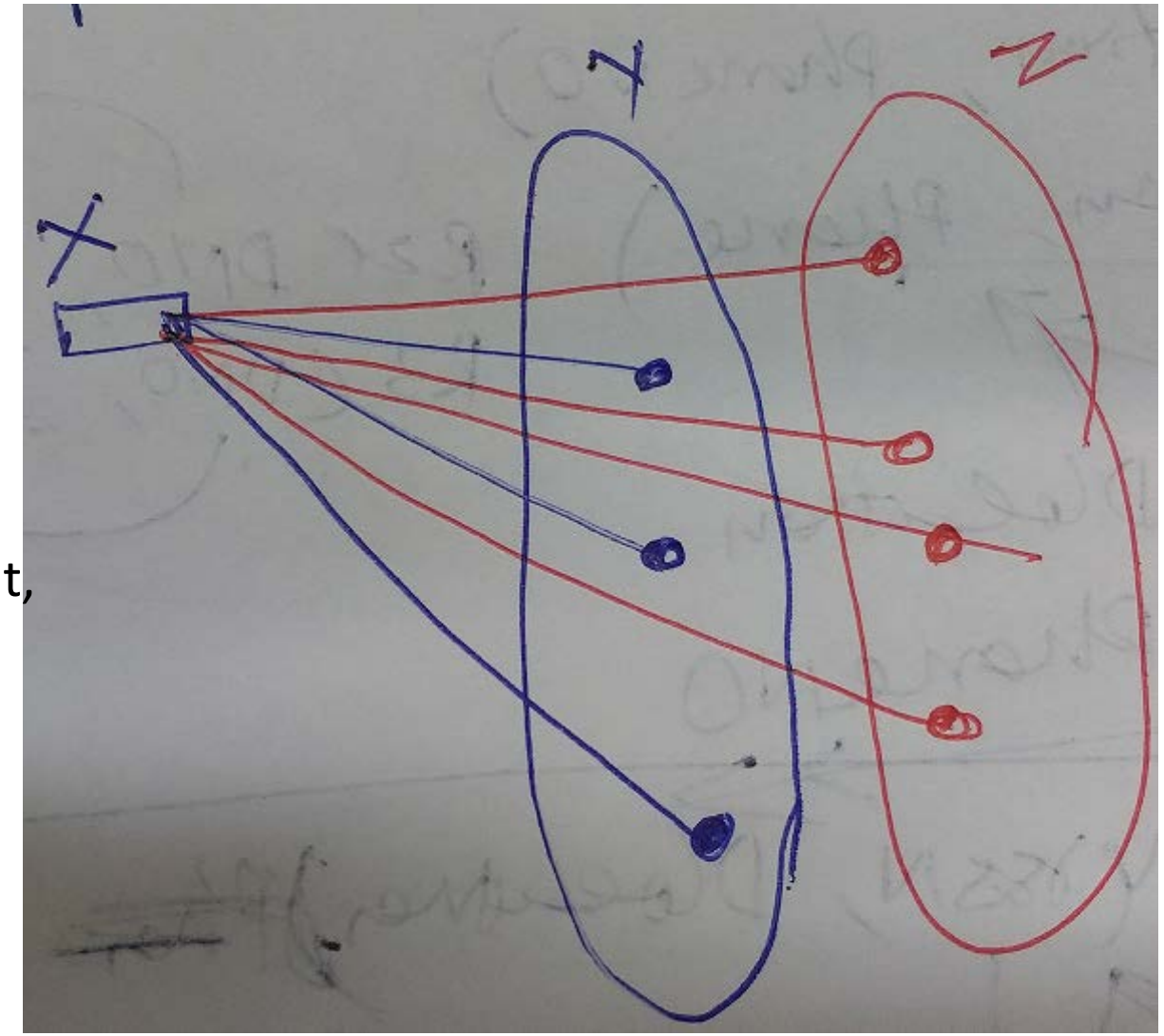
- MVD is represented as A -->> B, this means for a distinct value of A, you have set of values of B; and read as "**A multi-determines B**"

- Here determined is set (rather than a single value, as in the case of FD)

Multi Value Dependencies or MVDs

MVD phenomenon

- When X multi-value determines Y, and
- X also multi-value determines another attribute set Z, and
- Y and Z are independent, then you have X -->> Y and X -->> Z.
- Note that it may not be problem when there is only one MVD?



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

Combine and Split Rule do not apply to MVD

- ABC -->> X and ABC -->> Y,
  does not mean ABC -->> XY, and
- ABC -->> XY,
  does not mean ABC -->> X

- Example: Consider example from book [1]
  MovieActor(actor, street, city, title, year)

  MVDs:
  actor -->> {street, city}, and
  actor -->> {title, year}, and

FD is special case of MVD

- FD is special case of MVD, when determined set is singleton then it is FD

- **ssn → fname**, is also a MVD, however determined set is singleton

4NF based on MVDs

- So, R(UUID, email, phone) with following MVDs
  UUID -->> email
  UUID -->> phone

- Is in BCNF still has anomalies. We need to define a higher normal form - 4NF

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]

- Also, note that saying that relation is in 4NF, only if *it is in BCNF and there is no MVD is incorrect* (as FDs are MVDs)

- Definition of 4NF definition covers BCNF.

4NF decomposition

- If find any MVD X -->> Y in R violating 4NF requirement; decompose as following –

  R1(XY), and R2(R-R1UX)

4NF based on MVDs

- The relation R(UID, email, phone) with following MVDs: UID -->> email and UID -->> phone is not in 4NF because here UID is not the key; and MVDs are not trivial.

R decomposed using the 4NF algo

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

UID -->> email

UID	Phone
101	91011
101	91015

UID -->> Phone

MVDs in both decomposed relations are trivial

4NF Decomposition another example

- Applies(StudID, KnowledgeArea, Company)

StudID	StudentKA	Company
101	DBMS	Sapient
101	Algorithm	Sapient
101	DBMS	DirectI
101	Algorithm	DirectI
102	Communication	Qualcomm
102	Networking	Qualcomm

And you know what is the decomposition?

Relation decomposed based on MVD, both relations are in 4NF

StudID	StudentKA
101	DBMS
101	Algorithm
102	Communication
102	Networking

StudID -->> StudentKA

StudID	Company
101	Sapient
101	DirectI
102	Qualcomm

StudID -->> Company

MVDs in both decomposed relations are trivial

Note

- Every Relation that is 4NF is also in Boyce Codd Normal Form
- Every Relation that is BCNF is also in 3NF
- Every relation that is 3NF is also in 2NF, and
- Every relation that is in 2NF is also in 1NF

Higher normal forms and 5NF

- Though, 4NF is the highest form for most (all) practical purposes.
- However researchers define a most general Normal Form
- ==> You can skip rest of presentation; I am including this for the shake of completeness. Not included in exams.

Join Dependencies

- If you can have lossless decomposition of a relation R into R1 and R2, then relation R said to have a join dependency, represented as follows: ***{R1,R2}**

- You could 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)}**

- **You can have lossless decomposition of a relation only if it has join dependency**

Join Dependencies

- However such JD are detected through FDs and MVDs, which are more easy to find than discovering Join Dependencies

Join Dependencies

- In general, if a relation R is decomposed into Relation R1, R2, .. Rn, and
  if join of R1, R2, .. Rn is equal to R,

  then we say that following JD exists
  *{R1, R2, … Rn}

- And then we can have lossless decomposition of R into R1, R2, … Rn.

MVD/FDs are special cases of JD

- MVD is a special case JD, as

  A Relation R, has JD *{AB, R – B U A},
  when there is non-trivial MVD A -->> B, or we can say

  A -->> B in R implies JD ***{AB, R – B U A}**

- You also have JD in presence of FDs
  Any FD X →Y in R, has a join dependency
  ***{XY, R - Y U X }**

5NF

- A relation R is in 5NF- also called Projection Join Normal Form (PJNF)
  - When every non trivial join dependency JD (R1, R2, … Rn), every Ri is super key of R
- Suppose you have relation
  - EMP(SSN, ENAME, BDATE, SUPERSSN, SALARY), has a join dependency
    *{EMP1(SSN, ENAME, BDATE),
    EMP2(SSN, SUPERSSN, SALARY)}
  - every relation forms a super-key
  - Such JD are acceptable in in 5NF.

*JD is trivial if one of relation Ri in JD(R1, R2, .. Rn) is equal to R

5NF

- Only acceptable JDs are
  - Either trivial JD, or
  - JDs implied by candidate key, that means each of decomposed relation is super-key of original relation.

- With this understanding, hopefully, all the relations that we have claimed here in BCNF or 4NF are also in 5NF.

Join dependencies and decomposition

- Basically JDs take care of all FDs and MVDs;

  you will have JD *(AB, BC) on R(ABC) only when there is FD A → B and B → C, and

  you will have JD *(AB, AC) on R(ABC) only when there MVD A -->> B

- However JDs are difficult to discover, therefore we use theories of FDs and MVDs and JD based decomposition is rarely used in practice.

Join dependencies and decomposition

- Sometimes, we may not have binary join dependency on a relation (i.e. can be losslessly decomposed into two relations), but can be losslessly decomposed into more than two relations.

- Next is example of such typical JD

A typical JD

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

- Considering Supply-Part-Project schema and suppose we have a special constraint,
- if
  - a Supplier supplies S1 supplies part P1,
  - Part P1 is used in Project J1
  - S1 supplies to project J1
- Then
  - it is also true that S1 supplies P1 to project J1
- to record this fact we need to add a tuple (highlighted)

A typical JD

- **3-Decomposable relation SPJ**

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

S#	*P#*
S1	P1
S1	P2
S2	P1

P#	*J#*
P1	J2
P2	J1
P1	J1

S#	*J#*
S1	J2
S1	J1
S2	J1

- SP and PJ are joined over P#

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

- Then SJ is joined over J#S#

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

What is the problem in relation SPJ?

- If tuples (s1, p1, j2), (s2, p1,j1), (s1, p2,j1) appear in SPJ, then tuple (s1,p1,j1) must also exist.

- Ensuring this kind of constraint is difficult, and hence detecting JDs

Sources/References

[1] Database Systems: The Complete Book, by
Hector G. Molina, Jeff Ullman, and Jennifer Widom
Pearson Education, India
Home Page: http://infolab.stanford.edu/~ullman/dscb.html

[2] Elmasir/Navathe