# Module 9

# Design With Normal Forms





## Objectives

- Learn some of the algorithms which can be used to decompose a relation into normal forms and which lead to good decompositions

## 95.305

### Topics



## Decomposition

- How do we come up with a good set of relational tables?
- General strategy: Decompose tables that violate a normal form until all tables are in BCNF, or at least 3NF

- How do we get an initial set of tables?
- Put everything in one big table and start decomposing from there

## Decomposition of Universal Relation

- Assumes design process starts form a single universal relation

  R = {A1, A2, ... ,An}

  that includes all the attributes of the database (assumes all attribute names are unique)
- A set F of functional dependencies is specified by the designers



## ...Decomposition of Universal Relation

- The universal relation is decomposed, using the functional dependencies, into a set of relation schemas

  D = {R1, R2, ... ,Rm}

  that become the database schema

- D is called the decomposition of R

## ...Decomposition of Universal Relation

- What should be the characteristics of the decomposition D = {R1, R2, ... ,Rm}?
- Attribute Preservation: no attribute should be lost (R = Union of R1, ..., Rm)
- Each Ri should be in BCNF, or at least 3rd normal form
- Is this enough?



## Example Decomposition

EMP_PROJ

| SSN | PNUMBER | HOURS | ENAME | PNAME | PLOCATION |
|---|---|---|---|---|---|

EMP_LOC

| ENAME | PLOCATION |
|---|---|

(BCNF)

EMP_PROJ1

| SSN | PNUMBER | HOURS | PNAME | PLOCATION |
|---|---|---|---|---|

(not BCNF)

- Consider decomposing EMP_PROJ into two separate relations EMP_LOC and EMP_PROJ1
- EMP_LOC means employee name ENAME works on some project a location PLOCATION
- EMP_PROJ1 means employee SSN works on project PNUMBER for HOURS at PLOCATION
- Is this a good decomposition? (We know it's not)

## Does this lead to a good decomposition?

EMP_PROJ

| SSN | PNUMBER | HOURS | ENAME | PNAME | PLOCATION |
|---|---|---|---|---|---|

EMP_LOC (BCNF)

| ENAME | PLOCATION |
|---|---|

EMP_PROJ1 (not BCNF)

| SSN | PNUMBER | HOURS | PNAME | PLOCATION |
|---|---|---|---|---|

EMP_PROJ1A (BCNF)

| SSN | PNUMBER | HOURS |
|---|---|---|

EMP_PROJ1

| PNUMBER | PNAME | PLOCATION |
|---|---|---|



## EMP_PROJ Decomposition

**EMP_PROJ**

| SSN | PNUMBER | HOURS | ENAME | PNAME | PLOCATION |
|---|---|---|---|---|---|
| 123456789 | 1 | 32.5 | Smith, John | X | Bellaire |
| 123456789 | 2 | 7.5 | Smith, John | Y | Sugarland |
| 666884444 | 3 | 40 | Narayan, Ramesh | Z | Houston |
| 453453453 | 1 | 20 | English, Joyce | X | Bellaire |
| 453453453 | 2 | 20 | English, Joyce | Y | Sugarland |
| ... | ... | ... | ... | ... | ... |

**EMP_PROJ1**

| SSN | PNUMBER | HOURS | PNAME | PLOCATION |
|---|---|---|---|---|
| 123456789 | 1 | 32.5 | X | Bellaire |
| 123456789 | 2 | 7.5 | Y | Sugarland |
| 666884444 | 3 | 40 | Z | Houston |
| 453453453 | 1 | 20 | X | Bellaire |
| 453453453 | 2 | 20 | Y | Sugarland |
| ... | ... | ... | ... | ... |

**EMP_LOC**

| ENAME | PLOCATION |
|---|---|
| Smith, John | Bellaire |
| Smith, John | Sugarland |
| Narayan, Ramesh | Houston |
| English, Joyce | Bellaire |
| English, Joyce | Sugarland |
| ... | ... |

**Is this a good decomposition?**

## Attempt to recover EMP-PROJ info with a JOIN

**EMP_PROJ1**

| SSN | PNUMBER | HOURS | PNAME | PLOCATION |
|---|---|---|---|---|
| 123456789 | 1 | 32.5 | X | Bellaire |
| 123456789 | 2 | 7.5 | Y | Sugarland |
| 666884444 | 3 | 40 | Z | Houston |
| 453453453 | 1 | 20 | X | Bellaire |
| 453453453 | 2 | 20 | Y | Sugarland |
| ... | ... | ... | ... | ... |

**EMP_LOC**

| ENAME | PLOCATION |
|---|---|
| Smith, John | Bellaire |
| Smith, John | Sugarland |
| Narayan, Ramesh | Houston |
| English, Joyce | Bellaire |
| English, Joyce | Sugarland |
| ... | ... |

**Natural Join**

**Spurious Tuples**

| SSN | PNUMBER | HOURS | PNAME | PLOCATION | ENAME |
|---|---|---|---|---|---|
| 123456789 | 1 | 32.5 | X | Bellaire | Smith, John |
| 123456789 | 1 | 32.5 | X | Bellaire | English, Joyce |
| 123456789 | 2 | 7.5 | Y | Sugarland | Smith, John |
| 123456789 | 2 | 7.5 | Y | Sugarland | English, Joyce |
| 666884444 | 3 | 40 | Z | Houston | Narayan, Ramesesh |
| 453453453 | 1 | 20 | X | Bellaire | English, Joyce |
| 453453453 | 1 | 20 | X | Bellaire | Smith, John |
| 453453453 | 2 | 20 | Y | Sugarland | English, Joyce |
| 453453453 | 2 | 20 | Y | Sugarland | Smith, John |
| ... | ... | ... | ... | ... | ... |



## What went wrong?

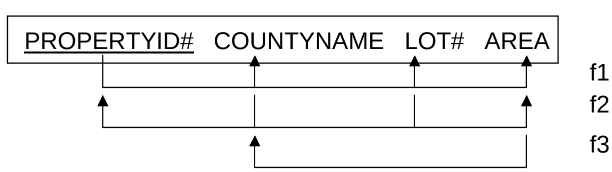
- **Good:**
  - -attributes were preserved
  - -individual tables did not violate normal forms
- **Bad**
  - -some decompositions are silly (emp-location)
  - -functional dependencies were not properly used to guide decomposition
  - -some functional dependencies may have “gotten lost”

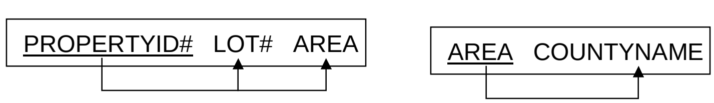
## Dependency Preservation

- **if X->Y appears in F, it would be nice if X->Y appeared in some Ri in the decomposition of R**
- **We want to preserve all functional dependencies because they are constraints on the database**
- **A functional dependency that appears in a single table is easy to check (no join required)**



## Example from [Elmasri & Navathe 12.5]




**What happed to f2:** COUNTYNAME, LOT# -> PROPERTYID#, AREA **?**

## Dependency Preservation

- if X->Y appears in F, it would be nice if X->Y appeared directly in some Ri in the decomposition of R
- Alternatively X->Y can be inferred from a dependency that appears in some Ri in the decomposition of R
- It is not necessary that the exact dependencies of F appear in individual relations, it is sufficient if those that do appear are equivalent to F



## Def'n: Projection of F on R

- The Projection of F on Ri is the set of dependencies X->Y in F+, such that Ri contains all the attributes of both X and Y
- A decomposition D = {R1, R2, ... ,Rm} is dependency preserving if

$$((\pi_F(R_1)) \cup \ldots \cup (\pi_F(R_m)))^+ = F^+$$

- If a decomposition is not dependency preserving, some dependency is <u>lost</u>

## Lost Dependencies

- To check whether a lost dependency X->Y holds we must join all the appropriate tables until all attributes of both X and Y appear in the resulting table
- Then we can check whether the data satisfies the dependency
- -not practical



## The Good News...

- It is always possible to find a dependency-preserving decomposition D with respect to F such that each table Ri in D is in 3NF

## Dependency Preserving Decomposition into 3NF

- **[Elmasri & Navathe] Algorithm 13.1**

**1) Find a minimal cover G of F**

**2) For each left-hand side X of a dependency in G**
**create a relation {X union A1 union A2 ... union Am}**
**in D where**
**X ->A1, X->A2,... X->Am**
**are the only dependencies in G with left-hand side X**

**3) Place any remaining attributes in a single relation to ensure attribute preservation**



## Minimal Cover G of F

- **Two sets of functional dependencies G and F are equivalent if G+ = F+**

- **A set of functional dependencies G is minimal if**

  **-every dependency in G as a single attribute for its right-hand side**

  **-We cannot replace any X->A in G with Y->A, where Y is a subset of X, and yield a set equivalent to F**

  **- We cannot remove any dependency from G and yield a set equivalent to F**

## Finding a Minimal Cover G of F

- [Elmasri & Navathe] Algorithm 13.1a

Wrong

```
1) G := F;
2) Replace each X->A1,A2,...,An in G by
   X->A1, X->A2, ... ,X->An;
3) For each X->A in G
       For each attribute B in X
         {compute X+ with respect to
             ( (G-(X->A)) UNION ((X-B)->A) );
           if X+ contains A, replace X->A with (X-B)->A in G };
4) For each remaining X->A in G
       {compute X+ with respect to (G- (X->A));
         if X+ contains A, remove X->A from G };
```



## Finding a Minimal Cover G of F

[Helman, P., “The Science of Database Management”, Irwin, 1994]

```
1) G := F;
2) Replace each X->A1,A2,...,An in G by
   X->A1, X->A2, ... ,X->An;
3) For each X->A in G  { //find a subset of X to serve as LHS
       Z := X;
       For each attribute b in X {
          G’ := G - {Z->A} UNION {Z-b->A};
          if (G’+ = G+)
            {Z := Z-b; G := G’};
        }
   };
4) For each remaining X->A in G
       {compute X+ with respect to (G- (X->A));
          if X+ contains A, remove X->A from G };
```

## Determining whether F+ = G+

[Helman, P., “The Science of Database Management”, Irwin, 1994]

```
Boolean isEqual(F,G) {
    For (each X->Y in F) {
            if ( Y not in X+  of G) return false;
    }
    For (each X->Y in G) {
            if (Y not in X+ of F) retrun false;
    }
    return true;
}
```



## Computing X+ of F

[Helman, P., “The Science of Database Management”, Irwin, 1994]

```
closure(X, F) {
X_prev := {};
X_current := X;
while (X_current != X_prev) {
    X_prev := X_current;
    X_current := X_current UNION Z, where Y->Z is in F and
                                              and Y is a subset of X_current
  }
return X_current;
}
```

## Example

EMP_DEPT

| | | | | | | |
|---|---|---|---|---|---|---|
| ENAME | SSN | BDATE | ADDRESS | DNUMBER | DNAME | DMGRSSN |

- F = { SSN -> ENAME, BDATE, ADDRESS, DNUMBER
  SSN,ENAME ->BDATE,ADDRESS
  DNUMBER -> DNAME, DMGRSSN
  DNAME -> DMGRSSN }

---

- **Is F minimal?**

- **Try to find a minimal cover G of F**



- F = { SSN -> ENAME, BDATE, ADDRESS, DNUMBER
  SSN,ENAME ->BDATE,ADDRESS
  DNUMBER -> DNAME, DMGRSSN
  DNAME -> DMGRSSN }

**1) G := F;**

**2) Replace each X->A1,A2,...,An in G by
X->A1, X->A2, ... ,X->An;**

- G = { SSN -> ENAME
  SSN -> BDATE
  SSN -> ADDRESS
  SSN -> DNUMBER
  SSN,ENAME -> BDATE
  SSN,ENAME -> ADDRESS
  DNUMBER -> DNAME
  DNUMBER -> DMGRSSN
  DNAME -> DMGRSSN }

G = { SSN -> ENAME
SSN -> BDATE
SSN -> ADDRESS
SSN -> DNUMBER
SSN,~~ENAME~~ -> BDATE
SSN,~~ENAME~~ -> ADDRESS
DNUMBER -> DNAME
DNUMBER -> DMGRSSN
DNAME -> DMGRSSN }

**3) For each X->A in G { //find a subset of X to serve as LHS**

```
Z := X;
For each attribute b in X {
    G' := G - {Z->A} UNION {Z-b->A};
    if (G'+ = G+)
       {Z := Z-b; G := G'};
  }
};
```



G = { SSN -> ENAME
~~SSN -> BDATE~~
~~SSN -> ADDRESS~~
SSN -> DNUMBER
SSN,~~ENAME~~ -> BDATE
SSN,~~ENAME~~ -> ADDRESS
DNUMBER -> DNAME
~~DNUMBER -> DMGRSSN~~
DNAME -> DMGRSSN }

**4) For each remaining X->A in G**

```
{compute X+ with respect to (G- (X->A));
   if X+ contains A, remove X->A from G };
```

## Possible minimal cover of F

- F = { SSN -> ENAME, BDATE, ADDRESS, DNUMBER
  SSN,ENAME ->BDATE,ADDRESS
  DNUMBER -> DNAME, DMGRSSN
  DNAME -> DMGRSSN }

### Some Possible minimal Covers of F

{SSN -> ENAME
SSN -> DNUMBER
SSN -> ADDRESS
SSN-> BDATE
DNUMBER -> DNAME
DNAME -> DMGRSSN }



## Example

EMP_DEPT

| ENAME | SSN | BDATE | ADDRESS | DNUMBER | DNAME | DMGRSSN |
|---|---|---|---|---|---|---|

- F = { SSN -> ENAME, BDATE, ADDRESS, DNUMBER
  SSN,ENAME ->BDATE,ADDRESS
  DNUMBER -> DNAME, DMGRSSN
  DNAME -> DMGRSSN }

---

- **Find a dependency preserving 3NF decomposition of** EMP_DEPT

## ...Example

1) Find a minimal cover G of F

2) For each left-hand side X of a dependency in G
create a relation {X union A1 union A2 ... union Am}
in D where
X ->A1, X->A2,... X->Am
are the only dependencies in G with left-hand side X

3) Place any remaining attributes in a single relation to
ensure attribute preservation



## ...Example

1) Find a minimal cover G of F

G = {SSN -> ENAME
SSN -> DNUMBER
SSN -> ADDRESS
SSN-> BDATE
DNUMBER -> DNAME
DNAME -> DMGRSSN }

## ...Example

**2) For each left-hand side X of a dependency in G create a relation {X union A1 union A2 ... union Am} in D where**
**X ->A1, X->A2,... X->Am**
**are the only dependencies in G with left-hand side X**

G = {SSN -> ENAME
SSN -> DNUMBER
SSN -> ADDRESS
SSN-> BDATE
DNUMBER -> DNAME
DNAME -> DMGRSSN }

R1= {SSN, ENAME, DNUMBER,ADDRESS,BDATE}
R2= {DNUMBER, DNAME}
R3= {DNAME, DMGRSSN }



## ...Example

**3) Place any remaining attributes in a single relation to ensure attribute preservation**

R1= {SSN, ENAME, DNUMBER,ADDRESS,BDATE}
R2= {DNUMBER, DNAME}
R3= {DNAME, DMGRSSN }

**There are no extra attributes (not mentioned in any dependency)**

**This decomposition is in 3NF and preserves all dependencies**

- **Are these equivalent?**

R1= {SSN, ENAME, DNUMBER,ADDRESS}
R2= {SSN,ENAME, BDATE}
R3= {DNUMBER, DNAME}
R4= {DNAME, DMGRSSN }

R1= {SSN, ENAME, BDATE, ADDRESS, DNUMBER}
R2= {DNUMBER, DNAME, DMGRSSN}

R1= {SSN, ENAME, DNUMBER,ADDRESS,BDATE}
R2= {DNUMBER, DNAME}
R3= {DNAME, DMGRSSN }



- **What about the keys**

R1= {<u>SSN</u>, ENAME, DNUMBER,ADDRESS}
R2= {<u>SSN,ENAME</u>, BDATE}
R3= {<u>DNUMBER</u>, DNAME}
R4= {<u>DNAME</u>, DMGRSSN }

R1= {<u>SSN</u>, ENAME, BDATE, ADDRESS, DNUMBER}
R2= {<u>DNUMBER</u>, DNAME, DMGRSSN}

## Multi-valued Dependencies

- **Consider the situation of bank customers who possibly have multiple address and multiple accounts**

| LOANS | | | |
|---|---|---|---|
| LOAN_NO | CUSTOMER | STREET | CITY |
| 101 | Sue | Elgin | Ottawa |
| 101 | Sue | Eagleson | Kanata |
| 112 | Frank | Bank | Ottawa |



- **Suppose Sue takes out another loan (#113) we could add the following tuple to the LOANS table**

<113, Sue, Elgin, Ottawa>

| LOANS | | | |
|---|---|---|---|
| LOAN_NO | CUSTOMER | STREET | CITY |
| 101 | Sue | Elgin | Ottawa |
| 101 | Sue | Eagleson | Kanata |
| 112 | Frank | Bank | Ottawa |
| 113 | Sue | Elgin | Ottawa |

- **Problem: Which of Sue’s addresses do we enter in the table**

- **Solution 1) add another tuple**

| LOANS | | | |
|---|---|---|---|
| LOAN_NO | CUSTOMER | STREET | CITY |
| 101 | Sue | Elgin | Ottawa |
| 101 | Sue | Eagleson | Kanata |
| 112 | Frank | Bank | Ottawa |
| 113 | Sue | Elgin | Ottawa |
| 113 | Sue | Eagleson | Kanata |



- **Solution 2) Decompose the relation**

| LOANS | |
|---|---|
| LOAN_NO | CUSTOMER |
| 101 | Sue |
| 112 | Frank |
| 113 | Sue |

| ADDRESS | | |
|---|---|---|
| CUSTOMER | STREET | CITY |
| Sue | Elgin | Ottawa |
| Sue | Eagleson | Kanata |
| Frank | Bank | Ottawa |

- **Multi-valued dependencies are a consequence of First Normal Form -attributes cannot be multi-valued**

- **If we do have two multi-valued attributes in a relation (e.g. loan, address) we have to repeat every value of one with every value of the other if we want to keep things consistent**

- **A Multi-valued dependency is a constraint which says that, in effect, that <u>each</u> loan must appear with <u>each</u> address**



- **Couldn't we just require**

  CUSTOMER -> ADDRESS
  CUSTOMER -> LOAN

- **The multi-valued dependency**

  CUSTOMER ->-> ADDRESS

  **does not rule out the possibility of multiple addresses, instead specifies that if a tuple contains one address, other tuples may need to be added with the other**



## Def'n Multi-valued Dependency

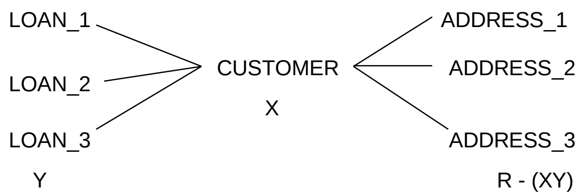
- **For a relation** r(R) **with attribute subsets** X **and** Y**, the multi-valued dependency** X->-> Y **requires that if two tuples** t1 **and** t2 **exist with** t1[X] = t2[X] **then two tuples** t3 **and** t4 **must also exist with**

  t3[X] = t4[X] = t1[X] = t2[X]
  t3[Y] = t1[Y] and t4[Y] = t2[Y]
  t3[R-(XY)] = t2[R-(XY)] and t4[R-(XY)] = t1[R-(XY)]

  **(**t1, t2, t3, t4 **need not be distinct)**

## Def’n Multi-valued Dependency

- **If** t1 **and** t2 **exist with** t1[X] = t2[X] **then** t3 **and** t4 **must exist with**
  t3[X] = t4[X] = t1[X] = t2[X]
  t3[Y] = t1[Y] and t4[Y] = t2[Y]
  t3[R-(XY)] = t2[R-(XY)] and t4[R-(XY)] = t1[R-(XY)]

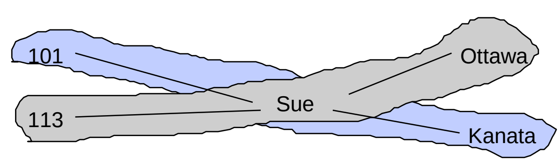




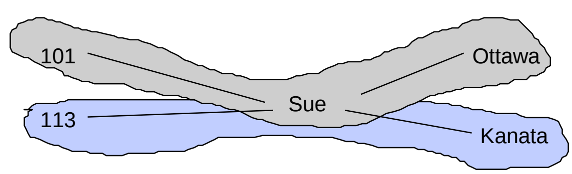

## Example

- **If** t1 **and** t2 **exist with** t1[X] = t2[X] **then** t3 **and** t4 **must exist with**
  t3[X] = t4[X] = t1[X] = t2[X]
  t3[Y] = t1[Y] and t4[Y] = t2[Y]
  t3[R-(XY)] = t2[R-(XY)] and t4[R-(XY)] = t1[R-(XY)]

Given tuples
<113, Sue, Ottawa>
<101, Sue, Kanata>
exist




Then so must
<113, Sue, Kanata>
<101, Sue, Ottawa>
exist



- **Does CUSTOMER ->-> ADDRESS hold here?**

**LOANS**

| | LOAN_NO | CUSTOMER | STREET | CITY | |
|---|---|---|---|---|---|
| | 101 | Sue | Elgin | Ottawa | t4 |
| t1 | 101 | Sue | Eagleson | Kanata | |
| | 112 | Frank | Bank | Ottawa | |
| t2 | 113 | Sue | Elgin | Ottawa | |

- **There is no** t3 **for which which agrees with** t1 **address and** t2 **loan -so no**

- **If** t1 **and** t2 **exist with** t1[X] = t2[X] **then** t3 **and** t4 **must exist with**
  t3[X] = t4[X] = t1[X] = t2[X]
  t3[Y] = t1[Y] and t4[Y] = t2[Y]
  t3[R-(XY)] = t2[R-(XY)] and t4[R-(XY)] = t1[R-(XY)]



- **Does CUSTOMER ->-> ADDRESS hold here?**

**LOANS**

| | LOAN_NO | CUSTOMER | STREET | CITY | |
|---|---|---|---|---|---|
| t1 | 101 | Sue | Elgin | Ottawa | |
| | 101 | Sue | Eagleson | Kanata | t4 |
| | 112 | Frank | Bank | Ottawa | |
| | 113 | Sue | Elgin | Ottawa | t3 |
| t2 | 113 | Sue | Eagleson | Kanata | |
| s1 | 111 | John | Riverside | Ottawa | s3 |
| s2 | 104 | John | Merivale | Nepean | s4 |

- **If** t1 **and** t2 **exist with** t1[X] = t2[X] **then** t3 **and** t4 **must exist with**
  t3[X] = t4[X] = t1[X] = t2[X]
  t3[Y] = t1[Y] and t4[Y] = t2[Y]
  t3[R-(XY)] = t2[R-(XY)] and t4[R-(XY)] = t1[R-(XY)]

- Whenever two independent 1:N relationships X:Y and X:Z are mixed in the same relation a multi-valued dependency could arise
- e.g. a customer's address is independent of the fact that they have a loan, however they can have several of each



## Trivial Multi-valued Dependencies

- If Y is a subset of X, X->->Y
- If X UNION Y = R, X->->Y
- These are called trivial because they hold in any legal relation R (and so don't specify any additional constraint

**LOANS**

| LOAN_NO | CUSTOMER | STREET | CITY |
|---|---|---|---|
| 101 | Sue | Elgin | Ottawa |
| 101 | Sue | Eagleson | Kanata |
| 112 | Frank | Bank | Ottawa |
| 113 | Sue | Elgin | Ottawa |
| 113 | Sue | Eagleson | Kanata |

- **Should we specify**
  CUSTOMER ->-> STREET,CITY **or**
  CUSTOMER ->-> LOAN
- **Does not matter because one implies the other**
- **If** X->->Y **then** X->-> R - X - Y



## Inference Rules for Functional and Multi-valued Dependencies

**For R=(A1,A2, ... ,An) and W, X, Y, Z all subsets of R, the following inference rules hold**

| | |
|---|---|
| **1) X->Y for any subset Y of X** | (reflexive) |
| **2) If X->Y then XZ -> YZ** | (augmentation) |
| **3) If X->Y and Y->Z, then X->Z** | (transitive) |
| **4) If X->->Y, then X->->(R - X - Y)** | (complementation) |
| **5) If X->->Y and Z is a subset of W then WX->->YZ** | (mv augmentation) |
| **6) If X->->Y, Y->->Z then X->->(Z-Y)** | (mv transitive) |
| **7)If X->Y, then X->->Y** | (replication) |
| **8)If X->-> Y and there is a W such that W** INTERSECT **Y is empty, W->Z, and Z is a subset of Y, then X->Z** | (coalescence) |

## Exercise

- **Let R=(A, B, C, G, H, I) with**
  **MVD = { A->-> B,**
  **B ->-> HI,**
  **CG -> H }**

- **Show that each of the following also hold**
  **A ->-> CGHI** (hint: complementation rule)
  **A ->-> HI** (hint: transitivity)
  **B -> H** (hint: coalescence)



## Do Functional Dependency normal forms help

- **Does CUSTOMER ->-> ADDRESS hold here?**

**LOANS**

| LOAN_NO | CUSTOMER | STREET | CITY |
|---|---|---|---|
| 101 | Sue | Elgin | Ottawa |
| 101 | Sue | Eagleson | Kanata |
| 112 | Frank | Bank | Ottawa |
| 113 | Sue | Elgin | Ottawa |

- **This table is in BCNF because no functional dependencies apply**
- **Still it is undesirable because of repeated information**
- **We need another kind of Normal Form**

## Fourth Normal Form

- **A relation schema R is in 4NF with respect to a set of dependencies F if, for every <u>nontrivial</u> MVD X->->Y in F+, X is a superkey of R**



## Forth Normal Form

**LOANS**

| LOAN_NO | CUSTOMER | STREET | CITY |
|---|---|---|---|
| 101 | Sue | Elgin | Ottawa |
| 101 | Sue | Eagleson | Kanata |
| 112 | Frank | Bank | Ottawa |
| 113 | Sue | Elgin | Ottawa |
| 113 | Sue | Eagleson | Kanata |

F = { CUSTOMER->->STREET,CITY
CUSTOMER->->LOAN_NO }

- **Violates 4NF because** CUSTOMER **is not a superkey**

  (and CUSTOMER->->LOAN_NO is non-trivial)

## Decomposition to 4NF

R - X - Y　　X　　Y

| LOANS | | | |
|---|---|---|---|
| LOAN_NO | CUSTOMER | STREET | CITY |
| 101 | Sue | Elgin | Ottawa |
| 101 | Sue | Eagleson | Kanata |
| 112 | Frank | Bank | Ottawa |
| 113 | Sue | Elgin | Ottawa |
| 113 | Sue | Eagleson | Kanata |

**X->->Y in F**
**X->Y not in F**

F = { CUSTOMER->->STREET,CITY
CUSTOMER->->LOAN_NO }

X　　R - X - Y

| LOANS | |
|---|---|
| CUSTOMER | LOAN |
| Sue | 101 |
| Frank | 112 |
| Sue | 113 |

X　　Y

| ADDRESS | | |
|---|---|---|
| CUSTOMER | STREET | CITY |
| Sue | Elgin | Ottawa |
| Sue | Eagleson | Kanata |
| Frank | Bank | Ottawa |

**Note: in the decomposed tables both MVD' of F are trivial, hence 4NF**



- **Given a database design situation with functional and multi-valued dependencies, it is advantageous to find a decomposition that is**

  **4NF**
  **Lossless Join**
  **Dependency Preserving**

- **This is not always possible, the compromise would be to relax 4NF and go to BCNF or 3NF and lose some dependency preservation**

- **You never want to relax the lossless-join constraint**

## Join Dependencies

- It is not always possible to decompose a relation R into to relations R1 and R2 which is lossless join
- However it may be possible to decompose R into more than two relations R1, R2, ... ,Rn which is a lossless join decomposition
- These cases are rare and difficult to detect in practice



## Join Dependency

- A join dependency JD(R1, R2, ... ,Rn) specifies a constraint on instances r(R) that every legal instance r(R) should have a lossless join decomposition into R1, R2, ... ,Rn.
- That is,

  JOIN( $\pi_{\langle R1 \rangle}(r)$, $\pi_{\langle R2 \rangle}(r)$, ... , $\pi_{\langle Rn \rangle}(r)$ ) = r

## Fifth Normal Form (Project Normal Form)

- A relation schema R is in 5NF with respect to a set F of functional, multi-valued, and join dependencies if, for every nontrivial join dependency JD(R1, R2, ... ,Rn) implied by F, every Ri is a superkey of R

- Current practical database design does not pay much attention to this normal form