Normal Form Design Algorithms

Objectives

- **Learn some of the algorithms used in the theory of normal form decomposition.**

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?

- Idea: Put everything in one big table and start decomposing from there

Decomposition of Universal Relation

- Assumes design process starts from a single universal relation

  $$R = \{A_1, A_2, \ldots ,A_n\}$$

  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

  $$D = \{R_1, R_2, \ldots, R_m\}$$

  that become the database schema

- D is called the decomposition of R

...Decomposition of Universal Relation

- What should be the characteristics of the decomposition $D = \{R_1, R_2, \ldots ,R_m\}$?
- Attribute Preservation: no attribute should be lost $R = R_1 \cup R_2 \cup \ldots \cup R_m$
- Each R_i should be in BCNF, or at least 3NF
- 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

ENAME	PLOCATION

(BCNF)

EMP_PROJ1

SSN	PNUMBER	HOURS	PNAME	PLOCATION

(not BCNF, violates 2NF)

EMP_PROJ1A

SSN	PNUMBER	HOURS

(BCNF)

EMP_PROJ1

PNUMBER	PNAME	PLOCATION

(BCNF)

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 maps to (or projects onto) some R_i 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)

Def'n: Projection of F on R

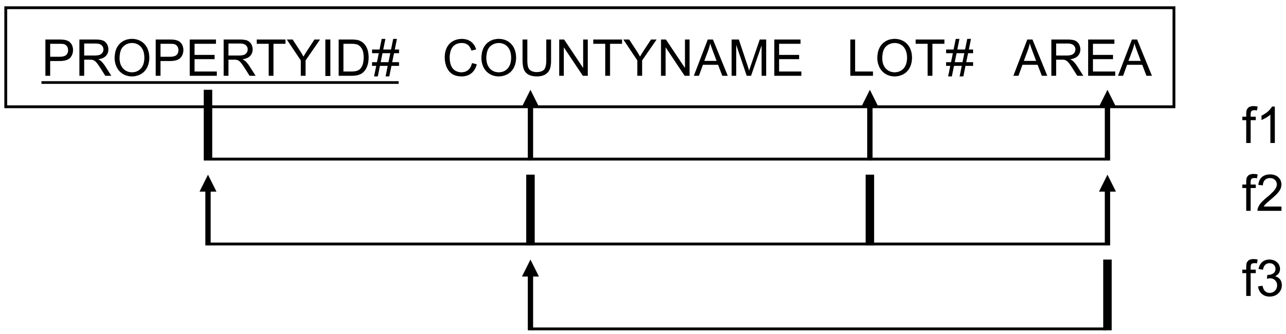
- The **Projection** $\pi_F(R_i)$ of F on R_i is the set of dependencies X->Y in F^+, such that R_i contains all the attributes of both X and Y

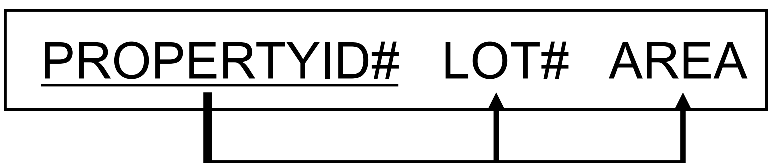
- A decomposition D = $\{R_1, R_2, \ldots, R_m\}$ is **Dependency Preserving** if

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

- If a decomposition is not dependency preserving, some dependency has been lost.

Example from [Elmasri & Navathe]





Exercise: Determine, and prove, whether this is, or is not, a dependency preserving decomposition.

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 R_i in the decomposition of R.

- Alternatively it is enough if X->Y can be inferred from those dependencies that do appear with the various R_i 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.

Lost Dependencies

- To check within data 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
- So a design that is not dependency preserving can have repetition that appears across several tables. The repetition will not be evident until the tables are joined.

The Good News...

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

- It is also always possible to find a dependency-preserving, lossless-join decomposition D with respect to F such that each table R_i in D is in 3NF (and the tables can be rejoined without creating spurious tuples.)

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 Set of Dependencies

- A set of functional dependencies is minimal if

  -every dependency has a single attribute for its right-hand side.

  -We cannot replace any X->A with Y->A, where Y is a proper subset of X, and yield a set equivalent to the original set.

  - We cannot remove any dependency from the set and yield a set equivalent to the original.

Equivalence and Minimal Cover

- **Two sets of functional dependencies G and F are equal if $G^+ = F^+$ (i.e. set equality: $G^+ \subseteq F^+$ and $F^+ \subseteq G^+$)**

- **A minimal cover F_m of F is a minimal set of functional dependencies equivalent to F.**

Finding a Minimal Cover G of F

- [Elmasri & Navathe] Algorithm 13.1a

WRONG

1) G = F //initialize

2) Replace each $X\text{->}A_1,A_2,\ldots,A_n$ in G by
   $X\text{->}A_1$, $X\text{->}A_2$, ... , $X\text{->}A_n$

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 $X\text{->}A_1,A_2,\ldots,A_n$ in G by
   $X\text{->}A_1$, $X\text{->}A_2$, ... ,$X\text{->}A_n$;

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} U {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 };
```

Finding a Minimal Cover G of F

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

1) G = F;

2) Replace each $X \rightarrow A_1, A_2, \ldots, A_n$ in G by
   $X \rightarrow A_1$, $X \rightarrow A_2$, ... , $X \rightarrow A_n$;

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} U {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 };

Interesting operations

Determining whether F==G (i.e. whether $F^+ = G^+$)

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

```
boolean equals(Set<FD> F, Set<FD> G) {
    For (each X->Y in F) {
            if ( Y not in X+ wrt. G) return false;
    }
    For (each X->Y in G) {
            if (Y not in X+ wrt. F) return false;
    }
    return true;
}
```

Determining whether F==G (i.e. whether $F^+ = G^+$)

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

```
boolean equals(Set<FD> F, Set<FD> G) {
   For (each X->Y in F) {
         if ( Y not in X+ wrt. G) return false;
   }
   For (each X->Y in G) {
         if (Y not in X+ wrt. F) return false;
   }
   return true;
}
```

Remaining Interesting operation

Computing Attribute Closure X^+ wrt. F

```
closure(Set<Attribute> X, Set<FD> F) {
xPrev = {};
xCurrent = X;  //application of reflexive rule
while (xCurrent != xPrev) {
    xPrev = xCurrent;
    //apply union rule
    xCurrent = xCurrent U Z, for any Y->Z in F with
                                        Y a subset of xCurrent
  }
return xCurrent;
}
```

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?**
- **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'};
        }
  }
```

Counter Example

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

Consider if its possible to remove SSN from SSN,ENAME -> BDATE.

{SSN,ENAME}$^+$ wrt G = {SSN,ENAME,BDATE,ADDRESS,DNUMBER}

{SSN,ENAME}$^+$ wrt G' = {SSN,ENAME,BDATE,ADDRESS,DNUMBER}

ENAME$^+$ wrt G' = {ENAME,BDATE,ADDRESS}

ENAME$^+$ wrt G = {ENAME}

So G != G' and the proposed removal of SSN is not possible.

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'};
    }
}
```

Counter Example

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

Consider if its possible to remove ENAME from SSN,ENAME -> BDATE.

{SSN,ENAME}$^+$ wrt G = {SSN,ENAME,BDATE,ADDRESS,DNUMBER}
{SSN,ENAME}$^+$ wrt G' = {SSN,ENAME,BDATE,ADDRESS,DNUMBER}

SSN$^+$ wrt G' = {SSN,ENAME,BDATE,ADDRESS,DNUMBER}
SSN$^+$ wrt G = {SSN,ENAME,BDATE,ADDRESS,DNUMBER}
So G == G' and the proposed removal of ENAME is correct.

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 }

Minimal Cover 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: Dependency Preserving, 3NF Decomposition

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 A_1 union A_2 ... union A_m}
in D where
$X \to A_1$, $X \to A_2$,... $X \to A_m$
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 A_1 union A_2 ... union A_m}
in D where
X ->A_1, X->A2,... X->A_m
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= [<u>SSN</u>, ENAME, DNUMBER,ADDRESS,BDATE]

R2= [<u>DNUMBER</u>, DNAME]
R3= [<u>DNAME</u>, 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 (but is not guaranteed to provide a lossless rejoin of all the tables.)

Lossless-Join, Dependency Preserving, 3NF Decomposition

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 A_1 union A_2 ... union A_m}
in D where
$X \rightarrow A_1$, $X \rightarrow A_2$,... $X \rightarrow A_m$
are the only dependencies in G with left-hand side X

3) If none of the relations created in 2) contains a key of R then create one more table in D consisting of a key of R. (Note this takes care of any "unplaced" attributes as in step 3 of the previous algorithm.)

4) If any table R_i is subsumed by another R_j, then remove R_i from the decomposition. (i.e. if all the columns of table R_i appear in table R_j then remove table R_i.)

Lossless-Join, Dependency Preserving, 3NF Decomposition

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 A_1 union A_2 ... union A_m}
in D where
$X \text{->} A_1$, $X\text{->}A_2$,... $X\text{->}A_m$
are the only dependencies in G with left-hand side X

3) If no relation created in 2) contains a key of R then
create one more table in D consisting of a key of R.
(Note this takes care of any "unplaced" attributes as in
step 3 of the previous algorithm.)

4) If any table R_i is subsumed by another R_j, then remove
R_i from the decomposition. (i.e. if all the columns of
table R_i appear in table R_j then remove table R_j.)

Algorithm for Finding a Key of R wrt. dependencies F

[Elmasri and Navathe]

1) Key K=R
2) For each attribute A_i in K {

    if $(K-A_i)^+$ with respect to F contains

    all attributes of R then set $K=K-A_i$

  }

Note this finds one candidate key, not all of them, and not necessarily the one with the fewest attributes.

How would you modify this to find all candidate keys?

Comparing the Two Algorithms

F = {SSN -> Ename ,Address, Birthdate
PNo -> Pname, PLocation
SSN,PNo -> Ename ,Address, Birthdate, Pname, PLocation }

Algm. 1) Dependency Preserving, 3NF

<u>SSN</u>,Ename ,Address, Birthdate
<u>Pno</u>,Pname, PLocation

Algm. 2) Lossless-Join, Dependency Preserving, 3NF

<u>SSN</u>,Ename ,Address, Birthdate
<u>Pno</u>,Pname, PLocation
<u>SSN,PNo</u>

Algorithm to find All Keys of R wrt. dependencies F

[L.D. Nel]

```
keys(Set<Attribute> R , Set<FD> F){


  Set<Key>  superkeys;
  superkeys.add(R);
  return keys(superkeys, R, F);
}
```

Recursively find all candidate keys of R wrt. F

WARNING: intended for small examples only:
Exponential Time recursion

…Algorithm to find All Keys of R with dependencies F

[L.D. Nel] (For small examples only: Exponential Time Algm)

```
keys(Set<Key> K, Set<Attribute> R , Set<FD> F){
 //K = set of superkeys which are not necessarily minimal

  if(K={}) return {};   //basis case

  superkey k = K.remove(0);
   if(|k| = 1) return {k} union keys(K,R,F);
   Set<Key> newPossibleKeys;
   for (each Attribute a in k) {
        if( (k-a)+ wrt F contains R ) newPossibleKeys.add(k-a);
   }
   if(newPossibleKeys = {}) return {k} union keys(K,R,F);
   else return keys(K union newPossibleKeys, R, F);
}
```

…Algorithm to find All Keys of R with dependencies F

> Hi Dr. Nel,
>
> So it turns out there doesn't exist an algorithm to compute candidate keys which runs in polynomial time with respect to the number of attributes in the relation and the number of functional dependencies in the worst case. Lucchesi & Osborn (1978) showed that deciding whether an attribute is prime is NP-complete. The wikipedia link below shows a case in which 2n relations yield 2^n candidate keys.
> However, there exist several algorithms which are polynomial with respect to the number of candidate keys (with some interesting ones based on graphs, as in Saiedian & Spencer (1996)). A simple algorithm is presented on wikipedia (https://en.wikipedia.org/wiki/Candidate_key#Determining_candidate_keys) based on the algorithm given by Lucchesi & Osborn.
> I've implemented it in Rust (which can compile to asm.js and be hosted in the browser) at https://github.com/CL4PTP/dbapp (along with some minimal cover and closure calculations), but the algorithm is so simple it can be adapted to any framework.
>
> Best,
> Bence Meszaros (COMP 3005 fall 2018)