Schema Refinement and Normal Forms

The Evils of Redundancy

Redundancy is the root of several problems associated with relational schemas:

- redundant storage, insert/delete/update anomalies

Integrity constraints, in particular *functional dependencies*, can be used to identify schemas with such problems and to suggest refinements.

Main refinement technique: *decomposition* (replacing ABCD with, say, AB and BCD, or ACD and ABD).

Decomposition should be used judiciously:

- Is there reason to decompose a relation?
- What problems (if any) does the decomposition cause?

Example: Constraints on Entity Set

Consider relation obtained from Hourly_Emps:

- Hourly_Emps (*ssn*, *name*, *lot*, *rating*, *hrly_wages*, *hrs_worked*)

Notation: We will denote this relation schema by listing the attributes: SNLRWH

- This is really the *set* of attributes {S,N,L,R,W,H}.
- Sometimes, we will refer to all attributes of a relation by using the relation name. (e.g., Hourly_Emps for SNLRWH)

Some FDs on Hourly_Emps:

- *ssn* is the key: S $\longrightarrow$ SNLRWH
- *rating* determines *hrly_wages*: R $\longrightarrow$ W

➢ *suppose hourly_wages attribute is determined permissible by the rating attribute.i.e for a given rating value, there is only one hourly_wages value.*

Example (Contd.)

Problems due to R $\longrightarrow$ W :

- *Redundant Storage*
- *Update anomaly*: Can we change W in just the 1st tuple of SNLRWH?
- *Insertion anomaly*: What if we want to insert an employee and don't know the hourly wage for his rating?
- *Deletion anomaly*: If we delete all employees with rating 5, we lose the information about the wage for rating 5!

S	N	L	R	W	H
123-22-3666	Attishoo	48	8	10	40
231-31-5368	Smiley	22	8	10	30
131-24-3650	Smethurst	35	5	7	30
434-26-3751	Guldu	35	5	7	32
612-67-4134	Madayan	35	8	10	40

This FD (R$\rightarrow$W) can NOT be expressed in terms of the ER model. Only FDs that determine all attributes of a relation (i.e., key constraints) can be expressed in the ER model. Therefore we can NOT detect it when we considered Hourly_Emps as an entity set during ER Modeling.

This integrity constraint (IC) is an example of functional dependency.

Can null values help?

Null values cannot help eliminate redundant storage or update anomalies

The insertion anomaly in the previous page may be solved by using null values, null values cannot address all insertion anomalies: we cannot record the hourly wages for a rating unless there is an employee with that rating (primary key field cannot store null)

Null values can not solve deletion anomalies either

Use of Decompositions

Decomposing Hourly_Emps into two relations:

Wages

R	W
8	10
5	7

Hourly_Emps2

S	N	L	R	H
123-22-3666	Attishoo	48	8	40
231-31-5368	Smiley	22	8	30
131-24-3650	Smethurst	35	5	30
434-26-3751	Guldu	35	5	32
612-67-4134	Madayan	35	8	40

Will 2 smaller tables be better?

Are all problems solved?

Problems related to decomposition

Queries over the original relation may require us to join the decomposed relations. If such queries are common, the performance penalty of decomposing the relation may not be acceptable.

In this case we may choose to live with some of the problems of redundancy and not decompose the relation.

It is important to be aware of the potential problems caused by such residual redundancy in the design and to take steps to avoid them

A good DB designer should have a firm grasp of normal forms and what problems they do (or do not) alleviate, the technique of decomposition, and potential problems with decompositions.

Functional Dependencies (FDs)

A <u>functional dependency</u> X -> Y holds over relation R if, for every allowable instance *r* of R:

- $t1 \in r$, $t2 \in r$, $\pi_X(t1) = \pi_X(t2)$ implies $\pi_Y(t1) = \pi_Y(t2)$
- i.e., given two tuples in *r*, if the X values agree, then the Y values must also agree. (X and Y are *sets* of attributes.)

An FD is a statement about *all* allowable relations.

- Must be identified based on semantics of application.
- Given some allowable instance *r1* of R, we can check if it violates some FD *f*, but we cannot tell if *f* holds over R!

K is a candidate key for R means that K-> R

- However, K -> R does not require K to be *minimal*!

Reasoning About FDs

Given some FDs, we can usually infer additional FDs:

- *ssn -> did, did lot->* implies *ssn-> lot*

An FD *f* is *implied by* a set of FDs *F* if *f* holds whenever all FDs in *F* hold.

- F^+ = *closure of F* is the set of all FDs that are implied by *F*.

Armstrong's Axioms (X, Y, Z are sets of attributes):

- *Reflexivity*: If X-> Y, then Y -> X
- *Augmentation*: If X -> Y, then XZ -> YZ for any Z
- *Transitivity*: If X-> Y and Y $\subseteq$ Z, then X -> Z

These are *sound* and *complete* inference rules for FDs!

Reasoning About FDs (Contd.)

Couple of additional rules (that follow from AA):

- *Union*: If X-> Y and X -> Z, then X -> YZ
- *Decomposition*: If X -> YZ, then X-> Y and X -> Z
- *Pseudotransitivity:*

Example

Contracts (Cid, Sid, Jid, Did, Pid, Qty, Value)

Depts (Did, Budget, Report)

Suppliers (Sid, Address)

Parts (Pid, Name, Cost)

Projects (Jid, Mgr)

Contracts(*cid,sid,jid,did,pid,qty,value*), and:

- C is the key: C-> CSJDPQV
- Project purchases each part using single contract: JP -> C
- Dept purchases at most one part from a supplier: SD -> P

JP-> C, C -> CSJDPQV imply JP -> CSJDPQV

SD -> P implies SDJ-> JP

SDJ -> JP, JP -> CSJDPQV imply SDJ-> CSJDPQV

Reasoning About FDs (Contd.)

Computing the closure of a set of FDs can be expensive. (Size of closure is exponential in # attrs!)

Typically, we just want to check if a given FD *X ->Y* is in the closure of a set of FDs *F*. An efficient check:

- Compute *attribute closure* of X (denoted X^+) wrt *F:*
-
  ```
  closure = X;
  repeat until there is no change: {
          if there is an FD U → V in F such that U ⊆ closure,
              then set closure = closure ∪ V·
          }
  ```

Does F = {A -> B, B-> C, C D -> E } imply A-> E?

- i.e, is A -> E in the closure F^+ ? Equivalently, is E in A^+ ?

Apply algorithm to compute attribute closure for A^+

closure = A;
repeat until there is no change: {
    if there is an FD $U \rightarrow V$ in F s.t. $U \subseteq$ closure,
        then set closure = closure $\cup$ V
    }

		$U \rightarrow V$
closure = A	FD:	$A \rightarrow B$
closure = AB	FD:	$B \rightarrow C$
closure = ABC		

$A^+ = \{A,B,C\}$ and $E \notin A^+$, hence $A \rightarrow E \notin F^+$

Another Example

Given a relation R with 5 attributes ABCDE and the following FDs: $A \rightarrow B$, $BC \rightarrow E$, and $ED \rightarrow A$. We want to find all keys for R. Any key K for R must satisfy $K \rightarrow ABCDE$. There are 5 + 10 + 10 + 5 + 1 = 31 different possibilities.

Let's first try K = A; we need to show $A \rightarrow ABCDE$

closure = A;

$U \rightarrow V$

closure = A FD: $A \rightarrow B$

closure = AB

$A^+ = \{A,B\}$ and $ABCDE \not\subseteq A^+$, hence $A \rightarrow ABCDE \notin F^+$. In other words, A is not a key for R.

Now you try K = BCD; you need to show BCD $\rightarrow$ ABCDE

Show BCD $\rightarrow$ ABCDE

closure = BCD;

$U \rightarrow V$

closure = BCD   FD: $BC \rightarrow E$

closure = BCDE   FD: $ED \rightarrow A$

Closure=ABCDE

$BCD^+ = \{A,B,C,D,E\}$ and ABCDE $\in BCD^+$, hence BCD $\rightarrow$ABCDE $\in F^+$. In other words, BCD is a key for R.

You try K = ACD; you need to show ACD $\rightarrow$ ABCDE

You try K = CDE; you need to show CDE $\rightarrow$ ABCDE

Hence, there are three keys for R:
CDE, ACD, BCD

Key Notions for Normalization

Closure of an attribute(s)

Candidate Key Generation

Prime Attributes

Non-Prime Attributes

NORMAL FORMS

Normal Forms

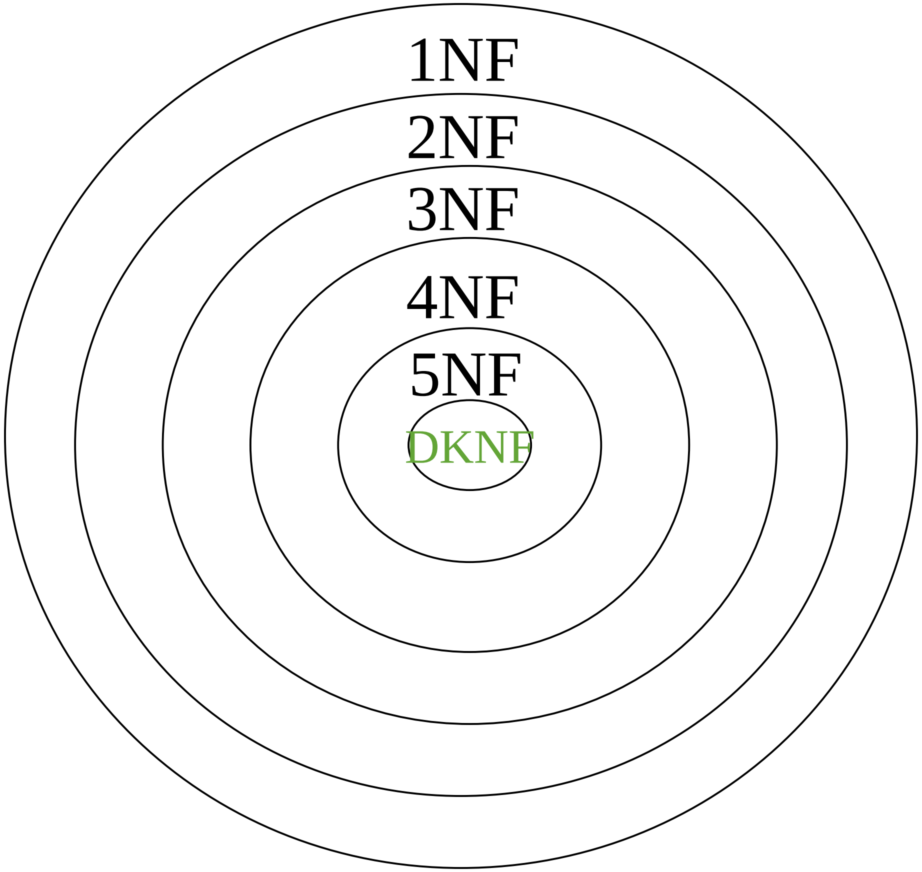
Returning to the issue of schema refinement, the first question to ask is whether any refinement is needed!

If a relation is in a certain *normal form* 1NF,2NF,BCNF, 3NF etc.), it is known that certain kinds of problems are avoided/minimized. This can be used to help us decide whether decomposing the relation will help.

Role of FDs in detecting redundancy:

- Consider a relation R with 3 attributes, ABC.
  - No FDs hold: There is no redundancy here.
  - Given A -> B: Several tuples could have the same A value, and if so, they'll all have the same B value!

Levels of Normalization



Each higher level is a subset of the lower level

First Normal Form (1NF)

A table is considered to be in 1NF if all the fields contain only scalar values (as opposed to list of values).

Example (Not 1NF)

ISBN	Title	AuName	AuPhone	PubName	PubPhone	Price
0-321-32132-1	Balloon	Sleepy, Snoopy, Grumpy	321-321-1111, 232-234-1234, 665-235-6532	Small House	714-000-0000	$34.00
0-55-123456-9	Main Street	Jones, Smith	123-333-3333, 654-223-3455	Small House	714-000-0000	$22.95
0-123-45678-0	Ulysses	Joyce	666-666-6666	Alpha Press	999-999-9999	$34.00
1-22-233700-0	Visual Basic	Roman	444-444-4444	Big House	123-456-7890	$25.00

Author and AuPhone columns are not scalar

1NF - Decomposition

1. Place all items that appear in the repeating group in a new table
2. Designate a primary key for each new table produced.
3. Duplicate in the new table the primary key of the table from which the repeating group was extracted or vice versa.

Example (1NF)

ISBN	Title	PubName	PubPhone	Price
0-321-32132-1	Balloon	Small House	714-000-0000	$34.00
0-55-123456-9	Main Street	Small House	714-000-0000	$22.95
0-123-45678-0	Ulysses	Alpha Press	999-999-9999	$34.00
1-22-233700-0	Visual Basic	Big House	123-456-7890	$25.00

ISBN	AuName	AuPhone
0-321-32132-1	Sleepy	321-321-1111
0-321-32132-1	Snoopy	232-234-1234
0-321-32132-1	Grumpy	665-235-6532
0-55-123456-9	Jones	123-333-3333
0-55-123456-9	Smith	654-223-3455
0-123-45678-0	Joyce	666-666-6666
1-22-233700-0	Roman	444-444-4444

Second Normal Form (2NF)

For a table to be in 2NF, there are two requirements

- The database is in first normal form
- All **nonkey** attributes in the table must be functionally dependent on the entire primary key / Every non-key attribute is fully dependent on each candidate key of the relation.
- It is not partially dependent on a candidate key.

Note: *Remember that we are dealing with non-key attributes*

Rule for Partial Dependency to exist: LHS should be a proper subset of Candidate key **and** RHS should be a non-prime/key attributes

Example 1 (Not 2NF)

Schema → {Title, PubId, AuId, Price, AuAddress}

1. Key → {Title, PubId, AuId}
2. {Title, PubId, AuID} → {Price}
3. {AuID} → {AuAddress}
4. AuAddress does not belong to a key
5. AuAddress functionally depends on AuId which is a subset of a key

Second Normal Form (2NF)

Example 2 (Not 2NF)

Schema → {studio, movie, budget, studio_city}

1. Key → {studio, movie}
2. {studio, movie} → {budget}
3. {studio} → {studio_city}
4. studio_city is not a part of a key
5. studio_city functionally depends on studio which is a proper subset of the key

Example 3 (Not 2NF)

Schema → {City, Street, HouseNumber, HouseColor, CityPopulation}

1. key → {City, Street, HouseNumber}
2. {City, Street, HouseNumber} → {HouseColor}
3. {City} → {CityPopulation}
4. CityPopulation does not belong to any key.
5. CityPopulation is functionally dependent on the City which is a proper subset of the key

2NF - Decomposition

1. If a data item is fully functionally dependent on only a part of the primary key, move that data item and that part of the primary key to a new table.
2. If other data items are functionally dependent on the same part of the key, place them in the new table also
3. Make the partial primary key copied from the original table the primary key for the new table. Place all items that appear in the repeating group in a new table

Example 1 (Convert to 2NF)

Old Scheme → {<u>Title, PubId, AuId</u>, Price, AuAddress}

New Scheme → {<u>Title, PubId, AuId</u>, Price}

New Scheme → {<u>AuId</u>, AuAddress}

2NF - Decomposition

Example 2 (Convert to 2NF)

Old Scheme → {Studio, Movie, Budget, StudioCity}

New Scheme → {Movie, Studio, Budget}

New Scheme → {Studio, City}

Example 3 (Convert to 2NF)

Old Scheme → {City, Street, HouseNumber, HouseColor, CityPopulation}

New Scheme → {City, Street, HouseNumber, HouseColor}

New Scheme → {City, CityPopulation}

Boyce-Codd Normal Form (BCNF)

Reln R with FDs *F* is in BCNF if, for all X -> A in F^+

- $A \in X$ (called a trivial FD), or
- X contains a key for R.

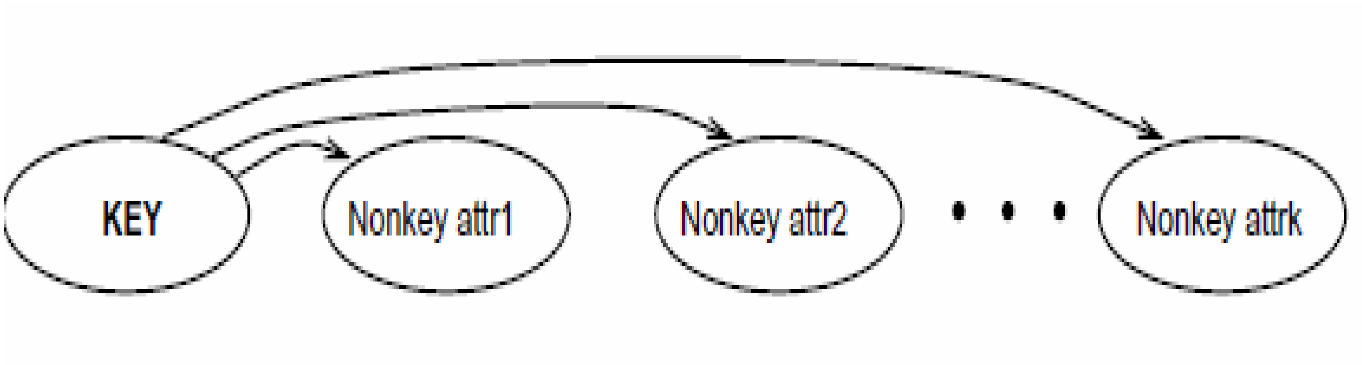
In other words, R is in BCNF if the only non-trivial FDs that hold over R are key constraints.

- No dependency in R that can be predicted using FDs alone.
- If we are shown two tuples that agree upon the X value, we cannot infer the A value in one tuple from the A value in the other.
- If example relation is in BCNF, the 2 tuples must be identical (since X is a key).

Rule for BCNF: LHS should be a Candidate key/Super Key

X	Y	A
x	y1	a
x	y2	?

BCNF



Decomposition of a Relation Scheme

Suppose that relation R contains attributes *A1 ... An.* A *decomposition* of R consists of replacing R by two or more relations such that:

- Each new relation scheme contains a subset of the attributes of R (and no attributes that do not appear in R), and
- Every attribute of R appears as an attribute of one of the new relations.

Intuitively, decomposing R means we will store instances of the relation schemes produced by the decomposition, instead of instances of R.

E.g., Can decompose SNLRWH into SNLRH and RW.

Example Decomposition

Decompositions should be used only when needed.

- SNLRWH has FDs S->SNLRWH and R -> W
- Second FD causes violation of 3NF; W values repeatedly associated with R values. Easiest way to fix this is to create a relation RW to store these associations, and to remove W from the main schema:
  - i.e., we decompose SNLRWH into SNLRH and RW

The information to be stored consists of SNLRWH tuples. If we just store the projections of these tuples onto SNLRH and RW, are there any potential problems that we should be aware of?

Problems with Decompositions

There are three potential problems to consider:

- Some queries become more expensive.
  - e.g., How much did an employee earn? (salary = W*H)
- Given instances of the decomposed relations, we may not be able to reconstruct the corresponding instance of the original relation!
- Checking some dependencies may require joining the instances of the decomposed relations.

Tradeoff: Must consider these issues vs. redundancy.

Lossless Join Decompositions

Decomposition of R into X and Y is *lossless-join* w.r.t. a set of FDs F if, for every instance *r* that satisfies F:

- $\pi_X(r) \bowtie \pi_Y(r) = r$

It is always true that $r \subseteq \pi_X(r) \bowtie \pi_Y(r)$

Definition extended to decomposition into 3 or more relations in a straightforward way.

It is essential that all decompositions used to deal with redundancy be lossless! (Avoids Problem (2).)

More on Lossless Join

The decomposition of R into X and Y is lossless-join wrt F if and only if the closure of F contains:

- $X \cap Y \rightarrow X$, or
- $X \cap Y \rightarrow Y$

In particular, the decomposition of R into UV and R - V is lossless-join if $U \rightarrow V$ holds over R.

Steps for Checking Lossless Join.

- The Union of Attributes of R1 and R2 must be equal to the attribute of R. Each attribute of R must be either in R1 or in R2.
  `Att(R1) U Att(R2) = Att(R)`
- The intersection of Attributes of R1 and R2 must not be NULL.
  `Att(R1) ∩ Att(R2) ≠ Φ`
- The common attribute must be a key for at least one relation (R1 or R2)
  `Att(R1) ∩ Att(R2) -> Att(R1) or Att(R1) ∩ Att(R2) -> Att(R2)`

A	B	C
1	2	3
4	5	6
7	2	8

A	B
1	2
4	5
7	2

B	C
2	3
5	6
2	8

A	B	C
1	2	3
4	5	6
7	2	8
1	2	8
7	2	3

Dependency Preserving Decomposition

Consider CSJDPQV, C is key, JP -> C and SD-> P.

- BCNF decomposition: CSJDQV and SDP
- Problem: Checking JP-> C requires a join!

Dependency preserving decomposition (Intuitive):

- If R is decomposed into X, Y and Z, and we enforce the FDs that hold on X, on Y and on Z, then all FDs that were given to hold on R must also hold. *(Avoids Problem (3).)*

Projection of set of FDs F: If R is decomposed into X, ... projection of F onto X (denoted F_X) is the set of FDs U ->V in F^+ (*closure of F*) such that U, V are in X.

Dependency Preserving Decompositions (Contd.)

Decomposition of R into X and Y is ***dependency preserving*** if $(F_X \text{ union } F_Y)^+ = F^+$

- i.e., if we consider only dependencies in the closure F^+ that can be checked in X without considering Y, and in Y without considering X, these imply all dependencies in F^+.

Important to consider F^+, not F, in this definition:

- ABC, A -> B, B-> C, C -> A, decomposed into AB and BC.
- Is this dependency preserving? Is C-> A preserved?????

Dependency preserving does not imply lossless join:

- ABC, A -> B, decomposed into AB and BC.

And vice-versa! (Example?)

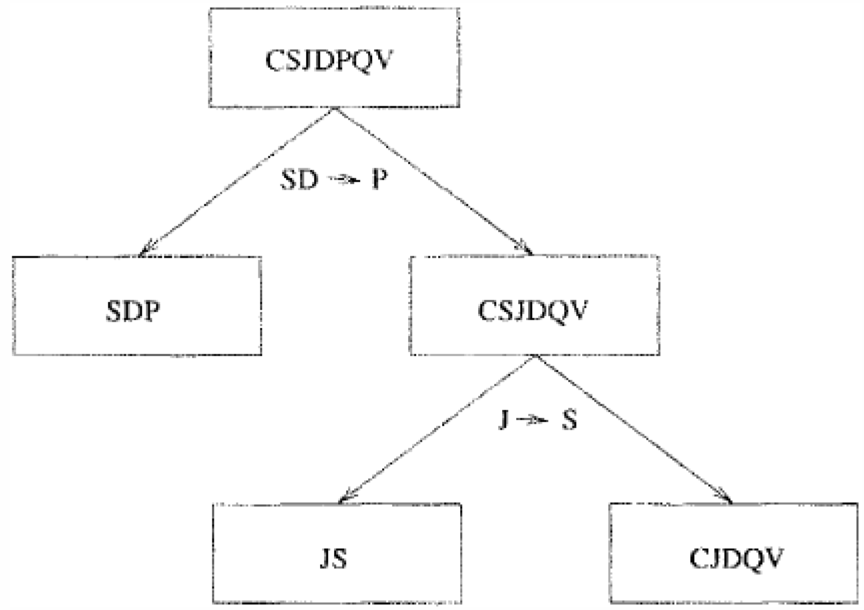
Dependency Preservation Problems

Decomposition into BCNF

Consider relation R with FDs F. If X -> Y violates BCNF, decompose R into R - Y and XY.

- Repeated application of this idea will give us a collection of relations that are in BCNF; lossless join decomposition, and guaranteed to terminate.
- e.g., CSJDPQV, key C, JP-> C, SD-> P, J -> S
- To deal with SD-> P, decompose into SDP, CSJDQV.
- To deal with J -> S, decompose CSJDQV into JS and CJDQV

In general, several dependencies may cause violation of BCNF. The order in which we \`\`deal with'' them could lead to very different sets of relations!


CSJDPQV
SD → P
SDP
CSJDQV
J → S
JS
CJDQV

BCNF and Dependency Preservation

In general, there may not be a dependency preserving decomposition into BCNF.

- e.g., CSZ, CS -> Z, Z -> C
- Can't decompose while preserving 1st FD; not in BCNF.

Similarly, decomposition of CSJDQV into SDP, JS and CJDQV is not dependency preserving (w.r.t. the FDs JP -> C, SD -> P and J -> S).

- However, it is a lossless join decomposition.
- In this case, adding JPC to the collection of relations gives us a dependency preserving decomposition.
  - JPC tuples stored only for checking FD! (*Redundancy!*)

Example of BCNF Decomposition

Original relation R and functional dependency F

R = (*branch_name, branch_city, assets,*
customer_name, loan_number, amount)

F = {*branch_name* $\rightarrow$ *assets branch_city*
loan_number $\rightarrow$ *amount branch_name* }

Key = {*loan_number, customer_name*}

Decomposition

- R_1 = (*branch_name, branch_city, assets*)
- R_2 = (*branch_name, customer_name, loan_number, amount*)
- R_3 = (*branch_name, loan_number, amount*)
- R_4 = (*customer_name, loan_number*)

Final decomposition

R_1, R_3, R_4

BCNF and Dependency Preservation

It is not always possible to get a BCNF decomposition that is dependency preserving

$R = (J, K, L)$
$F = \{JK \rightarrow L$
$\quad L \rightarrow K\}$
Two candidate keys = JK and JL

R is not in BCNF

Any decomposition of R will fail to preserve

$$JK \rightarrow L$$

This implies that testing for $JK \rightarrow L$ requires a join

Dependency Preservation

Example:

R = (*branch_name, customer_name, banker_name*)

F = {*banker_name* $\rightarrow$ *branch_name*

Customer_name branch_name $\rightarrow$ *banker_name* }

Decomposition

R1= (banker_name ,branch_name)

R2=(Customer_name, banker_name)

Dependency not preserved

Third Normal Form: Motivation

There are some situations where

- BCNF is not dependency preserving, and
- efficient checking for FD violation on updates is important

Solution: define a weaker normal form, called Third Normal Form (3NF)

- Allows some redundancy
- But functional dependencies can be checked on individual relations without computing a join.
- There is always a lossless-join, dependency-preserving decomposition into 3NF.

Third Normal Form (3NF)

Reln R with FDs *F* is in 3NF if, for $\in$all X A in

- A -> X (called a *trivial* FD), or
- X contains a key for R, or
- A is part of some key for R.

Minimality of a key is crucial in third condition above!

If R is in BCNF, obviously in 3NF.

If R is in 3NF, some redundancy is possible. It is a compromise, used when BCNF not achievable (e.g., no ``good'' decomp, or performance considerations).

- *Lossless-join, dependency-preserving decomposition of R into a collection of 3NF relations always possible.*

What Does 3NF Achieve?

If 3NF is violated by X -> A, one of the following holds:

- X is a subset of some key K
  - We store (X, A) pairs redundantly.
- X is not a proper subset of any key.
  - There is a chain of FDs K -> X -> A, which means that we cannot associate an X value with a K value unless we also associate an A value with an X value.

Figure 15.9 Partial Dependencies

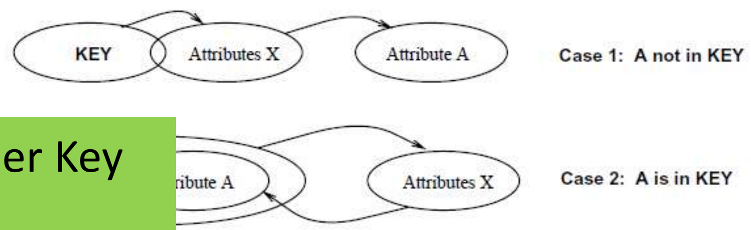


Figure 15.10 Transitive Dependencies

Rule for 3NF: LHS should be a Candidate key/Super Key **or** RHS should be a Prime Attribute

But: even if reln is in 3NF, these problems could arise.

- e.g., Reserves SBDC, S -> C, C -> S is in 3NF, but for each reservation of sailor S, same (S, C) pair is stored.

Thus, 3NF is indeed a compromise relative to BCNF.

3NF Example

Relation R:

- $R = (J, K, L)$
  $F = \{JK \rightarrow L, L \rightarrow K\}$
- Two candidate keys: JK and JL
- R is in 3NF

$JK \rightarrow L$	JK is a superkey
$L \rightarrow K$	K is contained in a candidate key

Redundancy in 3NF

There is some redundancy in this schema

Example of problems due to redundancy in 3NF

- $R = (J, K, L)$
  $F = \{JK \rightarrow L, L \rightarrow K\}$

J	L	K
j_1	l_1	k_1
j_2	l_1	k_1
j_3	l_1	k_1
null	l_2	k_2

- repetition of information (e.g., the relationship l_1, k_1)
- need to use null values (e.g., to represent the relationship l_2, k_2 where there is no corresponding value for J).

Testing for 3NF

Optimization: Need to check only FDs in *F*, need not check all FDs in F^+.

Use attribute closure to check for each dependency $\alpha \rightarrow \beta$, if α is a superkey.

If α is not a superkey, we have to verify if each attribute in β is contained in a candidate key of *R*

- this test is rather more expensive, since it involve finding candidate keys

Decomposition into 3NF

Obviously, the algorithm for lossless join decomp into BCNF can be used to obtain a lossless join decomp into 3NF (typically, can stop earlier).

Example

R = (*branch_name, customer_name, banker_name, Office_number*)

F = {*banker_name* $\rightarrow$ *branch_name office_number*

Customer_name branch_name $\rightarrow$ *banker_name* }

Decomposition

- R_1 = (*banker_name, branch_name, office_number*)
- R_2 = (*branch_name, customer_name, banker_name*)

Minimal Cover

A set of functional dependencies G covers another set of functional dependencies F, if every functional dependency in F can be inferred from G.

More formally, G covers F if $F^+ \subseteq G^+$. G is a minimal cover of F if G is the smallest set of functional dependencies that cover F.

Minimal Cover for a Set of FDs

Minimal cover G of FDs for a set of FDs F such that

- Closure of F = closure of G.
- Every dependency in g is of the form X-> A, where A is a Single attribute
- If we modify G by deleting an FD or by deleting attributes from an FD in G, the closure changes.

Intuitively, every FD in G is needed, and ``*as small as possible*'' in order to get the same closure as F.

e.g., A-> B, ABCD-> E, EF-> GH, ACDF-> EG has the following minimal cover:

- A-> B, ACD->E, EF->G and EF -> H

Algorithm for minimal cover

1. **Put the FDs in a standard form:** Obtain a collection G of equivalent FDs with a single attribute on the right side (using the decomposition axiom).
2. **Minimize the left side of each FD:** For each FD in G, check each attribute in the left side to see if it can be deleted while preserving equivalence to F^+.
3. **Delete redundant FDs:** Check each remaining FD in G to see if it can be deleted while preserving equivalence to F^+.

Dependency Preserving Decomposition into 3NF

Let R be a relation with a set F of FDs that is a minimal cover, and let R1 , R2 , ... , Rn be a lossless-join decomposition of R. For 1 < i < n, suppose that each Ri is in 3NF and let Fi denote the projection of F onto the attributes of Ri . Do the following:

Identify the set N of dependencies in F that is not preserved, that is, not included in the closure of the union of Fi s.

For each FD X → A in N, create a relation schema XA and add it to the decomposition of R.

For every relation schema that is a subset of some other relation schema, remove the smaller one.

The set of the remaining relation schemas is an almost final decomposition

This algorithm is almost correct, because it may be possible to compute a set of relations that do not contain the key of the original relation. If that is the case, you need to add a relation whose attributes form such a key.

Comparison of BCNF and 3NF

It is always possible to decompose a relation into a set of relations that are in 3NF such that:

- the decomposition is lossless
- the dependencies are preserved

It is always possible to decompose a relation into a set of relations that are in BCNF such that:

- the decomposition is lossless
- it may not be possible to preserve dependencies.

Summary of Schema Refinement

If a relation is in BCNF, it is free of redundancies that can be detected using FDs. Thus, trying to ensure that all relations are in BCNF is a good heuristic.

If a relation is not in BCNF, we can try to decompose it into a collection of BCNF relations.

- Must consider whether all FDs are preserved. If a lossless-join, dependency preserving decomposition into BCNF is not possible (or unsuitable, given typical queries), should consider decomposition into 3NF.
- Decompositions should be carried out and/or re-examined while keeping *performance requirements* in mind.

Multivalued Dependendencies

BCNF Relation with redundancy that is revealed by MVDs

course	*teacher*	*book*
Physics101	Green	Mechanics
Physics101	Green	Optics
Physics101	Brown	Mechanics
Physics101	Brown	Optics
Math301	Green	Mechanics
Math301	Green	Vectors
Math301	Green	Geometry

Let R be a relation schema and let X and Y be subsets of the attributes of R. The multivalued dependency X→→Y is said to hold over R if in every legal instance r of R, each value is associated with a set of Y values and this set is independent of the values in the other attributes

Definition of MVD

A *multivalued dependency* (MVD) on *R*, *X ->->Y* , says that if two tuples of *R* agree on all the attributes of *X*, then their components in *Y* may be swapped, and the result will be two tuples that are also in the relation.

i.e., for each value of *X*, the values of *Y* are independent of the values of *R*-*X*-*Y*.

Example

Drinkers(name, addr, phones, beersLiked)

A drinker's phones are independent of the beers they like.

- name->->phones and name ->->beersLiked.

Thus, each of a drinker's phones appears with each of the beers they like in all combinations.

This repetition is unlike FD redundancy.

- name->addr is the only FD.

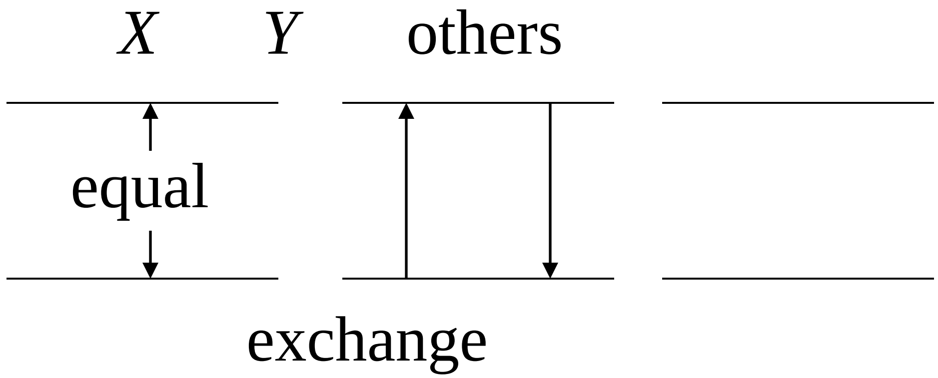
Tuples Implied by name->->phones

If we have tuples:

name	addr	phones	beersLiked
sue a	p1	b1	
sue a	p2	b2	
sue a	p2	b1	
sue a	p1	b2	

Then these tuples must also be in the relation.

Picture of MVD X ->->Y



MVD Rules

Every FD is an MVD (*promotion*).

- If *X ->Y*, then swapping *Y* 's between two tuples that agree on *X* doesn't change the tuples.
- Therefore, the "new" tuples are surely in the relation, and we know *X ->->Y*.

Splitting Doesn't Hold

Like FD's, we cannot generally split the left side of an MVD.

But unlike FD's, we cannot split the right side either --- sometimes you have to leave several attributes on the right side.

Example

Drinkers(name, areaCode, phone, beersLiked, manf)

A drinker can have several phones, with the number divided between areaCode and phone (last 7 digits).

A drinker can like several beers, each with its own manufacturer.

Example, Continued

Since the areaCode-phone combinations for a drinker are independent of the beersLiked-manf combinations, we expect that the following MVD's hold:

name ->-> areaCode phone

name ->-> beersLiked manf

Example Data

Here is possible data satisfying these MVD's:

name	areaCode	phone	beers Liked	manf
Sue	650	555-1111	Bud	A.B.
Sue	650	555-1111	WickedAle	Pete's
Sue	415	555-9999	Bud	A.B.
Sue	415	555-9999	WickedAle	Pete's

But we cannot swap area codes or phones by themselves. That is, neither name->->areaCode nor name->->phone holds for this relation.

Fourth Normal Form

The redundancy that comes from MVD’s is not removable by putting the database schema in BCNF.

There is a stronger normal form, called 4NF, that (intuitively) treats MVD’s as FD’s when it comes to decomposition, but not when determining keys of the relation.

4NF Definition

A relation *R* is in *4NF* if: whenever *X* ->->*Y* is a nontrivial MVD, then *X* is a superkey.

- *Nontrivial MVD* means that:
  1. *Y* is not a subset of *X*, and
  2. *X* and *Y* are not, together, all the attributes.
- Note that the definition of "superkey" still depends on FD's only.

BCNF Versus 4NF

Remember that every FD $X \rightarrow Y$ is also an MVD, $X \twoheadrightarrow Y$.

Thus, if R is in 4NF, it is certainly in BCNF.

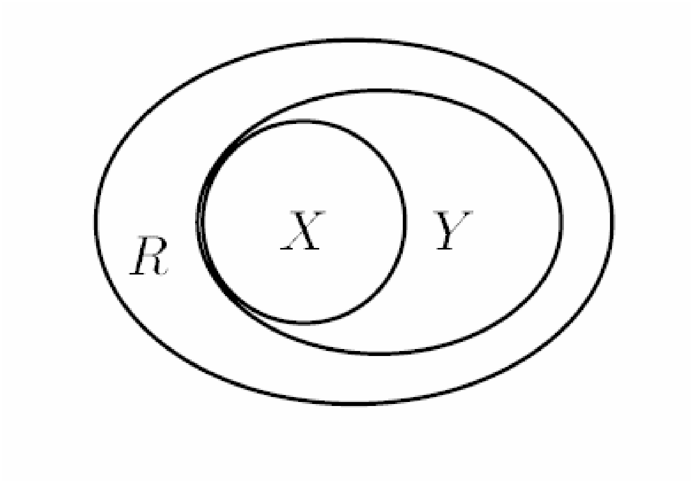
- Because any BCNF violation is a 4NF violation (after conversion to an MVD).

But R could be in BCNF and not 4NF, because MVD's are "invisible" to BCNF.

Decomposition and 4NF

If *X ->->Y* is a 4NF violation for relation *R*, we can decompjose *R* using the same technique as for BCNF.

1. *XY* is one of the decomposed relations.
2. All but X U (R-Y) is the other.



Example

Drinkers(<u>name</u>, addr, <u>phones</u>, <u>beersLiked</u>)

FD: name -> addr

MVD's: name ->-> phones

name ->-> beersLiked

Key is {name, phones, beersLiked}.

All dependencies violate 4NF.

Example, Continued

Decompose using name -> addr:

1. Drinkers1(name, addr)
   - In 4NF; only dependency is name -> addr.
2. Drinkers2(name, phones, beersLiked)
   - Not in 4NF. MVD's name ->-> phones and name ->-> beersLiked apply. No FD's, so all three attributes form the key.

Example: Decompose Drinkers2

Either MVD name ->-> phones or name ->-> beersLiked tells us to decompose to:

- Drinkers3(<u>name</u>, <u>phones</u>)
- Drinkers4(<u>name</u>, <u>beersLiked</u>)

If a relation schema is in BCNF, and at least one of its keys consists of a single attribute, it is also in 4NF.

Join Dependency

A join dependency (JD) $\bowtie$ {R1, R2…Rn) is said to hold over a relation R if R1 R2..Rn is a lossless-join decomposition of R

An MVD X→→Y over a relation R can be expressed as the join dependency{XY,X(R-Y)}

A relation R is in Fifth Normal Form if and only if every join dependency in R is implied by the candidate keys of R.

A relation decomposed into two relations must have lossless join Property, which ensures that no spurious or extra tuples are generated when relations are reunited through a natural join.

A relation R is in 5NF if and only if it satisfies the following conditions:

1. R should be already in 4NF.

2. It cannot be further non loss decomposed (join dependency).

Consider the above schema, with a case as “if a company makes a product and an agent is an agent for that company, then he always sells that product for the company”. Under these circumstances, the ACP table is shown as:

Table ACP

Agent	Company	Product
A1	PQR	Nut
A1	PQR	Bolt
A1	XYZ	Nut
A1	XYZ	Bolt
A2	PQR	Nut

The relation ACP is again decomposed into 3 relations.

Table R1

Agent	Company
A1	PQR
A1	XYZ
A2	PQR

Table R2

Agent	Product
A1	Nut
A1	Bolt
A2	Nut

Table R3

Company	Product
PQR	Nut
PQR	Bolt
XYZ	Nut
XYZ	Bolt

The result of the Natural Join of R1 and R3 over 'Company' and then the Natural Join of R13 and R2 over 'Agent'and 'Product' will be **Table ACP**.

Hence, in this example, all the redundancies are eliminated, and the decomposition of ACP is a lossless join decomposition. Therefore, the relation R1,R2,R3 is in 5NF as it does not violate the property of lossless join.

Problems to be Practiced

Closure of an Attribute

Candidate Key Generation

Prime Attributes

Non-Prime Attributes

Highest Normal Form

Decomposition of relations to the next higher normal form

Checking for lossy/lossless decomposition

Checking Dependency Preservation

Computing Canonical Cover/Minimal Cover of FDs

Decomposition of relations based on Canonical Cover