Normalization

CS 317/387

1

Limitations of DB Design Processes

- Provides a set of guidelines, does not result in a unique database schema
- Does not provide a way of evaluating alternative schemas
- Pitfalls:
  - Repetition of information
  - Inability to represent certain information
  - Loss of information
- Normalization theory provides a mechanism for analyzing and refining the schema produced by an E-R design

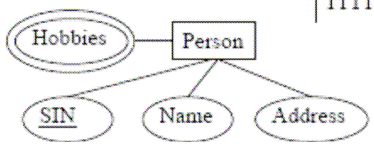
2

Redundancy and Other Problems

- Dependencies between attributes cause redundancy
  - Ex. All addresses in the same town have the same zip code
- Set valued attributes in the E-R diagram result in multiple rows in corresponding table
- Example: **`Person (PAN, Name, Address, Hobbies)`**
- A person entity with multiple hobbies yields multiple rows in table Person
  - Hence, the association between *Name* and *Address* for the same person is stored redundantly
- *PAN* is key of entity set, but (*PAN, Hobby*) is key of corresponding relation
- The relation Person can't describe people without hobbies

3

ER Model



SSN	*Name*	*Address*	*Hobby*
1111	Joe	123 Main	{biking, hiking}

Relational Model

SSN	*Name*	*Address*	*Hobby*
1111	Joe	123 Main	biking
1111	Joe	123 Main	hiking
.....			

Redundancy

4

Redundancy leads to anomalies

- **Update anomaly**: A change in *Address* must be made in several places
- **Deletion anomaly**: Suppose a person gives up all hobbies. Do we:
  - Set Hobby attribute to null? No, since *Hobby* is part of key
  - Delete the entire row? No, since we lose other information in the row
- **Insertion anomaly**: *Hobby* value must be supplied for any inserted row since *Hobby* is part of key

5

Solution: Decomposition

- **Solution**: use two relations to store Person information

Person1 (*PAN, Name, Address*)

Hobbies (*PAN, Hobby*)

- The decomposition is more general: people with hobbies can now be described
- No update anomalies:
  - Name and address stored once
  - A hobby can be separately supplied or deleted

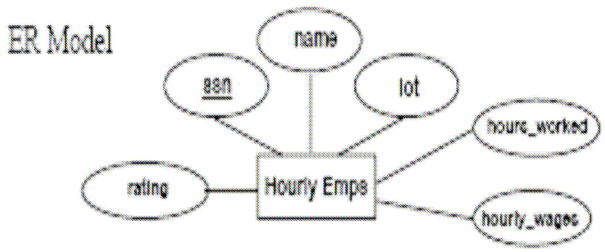
6

Normalization

- Result of E-R analysis need further refinement
- Appropriate decomposition can solve problems
- The underlying theory is referred to as *normalization theory* and is based on *functional dependencies* (and other kinds, like *multivalued dependencies*)

7

Example



- Hourly_Emps (*pan, name, lot, rating, hrly_wages*, *hrs_worked*)
- Some functional dependencies on Hourly_Emps:
- *pan* is the key: P -> PNLRWH
- *rating* determines *hrly_wages*: R -> W

Are there anomalies?

8

Functional Dependencies

- **Definition:** A *functional dependency* (FD) on a relation schema **R** is a constraint $\boldsymbol{X \rightarrow Y}$**,** where *X* and *Y* are subsets of attributes of **R.**

- **Definition**: An FD $\boldsymbol{X \rightarrow Y}$ is *satisfied* in an instance **r** of **R** if for every pair of tuples, *t* and s: if *t* and *s* agree on all attributes in *X* then they must agree on all attributes in *Y*.

- ***formally***: $\pi_X(t) = \pi_X(s) \Rightarrow \pi_y(t) = \pi_y(s)$ *w*here ($\pi_X(t)$ is the projection of tuple t onto the attributes X)

9

FDs

- **Definition**: A *constraint* on a relation schema R is a condition that has to be *satisfied in every allowable instance of* R.
- FDs must be identified based on semantics of application.

- Given a particular allowable instance *r1* of R, we can check if it violates some FD *f*, but we cannot tell if *f* holds over the schema R!

- A key constraint is a special kind of functional dependency: all attributes of relation occur on the right-hand side of the FD:
- $PAN \rightarrow PAN, Name, Address$

10

More examples

- *Address* → *ZipCode*
  - Powai zip is 400076
- *ArtistName* → *BirthYear*
  - Picasso was born in 1881
- *VIN* → *Manufacturer*, *Engine type, …*
  - VIN (vehicle information number) encodes all the information about manufacturer etc
- *Author, Title* → *PublDate*
  - Shakespeare's Hamlet published in 1600

11

Example from Specifications

- Brokerage firm allows multiple clients to share an account, but each account is managed from a single office and a client can have no more than one account in an office
- HasAccount (*AcctNum, ClientId, OfficeId*)
- Keys are (*ClientId, OfficeId*), (*AcctNum, ClientId*)
- *ClientId, OfficeId* → *AcctNum*
- *AcctNum* → *OfficeId*
- Thus, attribute values need not depend only on key values

12

Entailment, Closure and Equivalence

- **Definition**: If F is a set of FDs on schema **R** and f is another FD on **R**, then F *entails* f if every instance **r** of **R** that satisfies every FD in F also satisfies f
  - Ex: $F = \{A \rightarrow B, B \rightarrow C\}$ and f is $A \rightarrow C$
  - If *Streetaddr* $\rightarrow$ *Town* and *Town* $\rightarrow$ *Zip* then *Streetaddr* $\rightarrow$ *Zip*
  - *Aka "Follows from"*
- **Definition**: The *closure* of F, denoted $F+$, is the set of all FDs entailed by F
- **Definition**: F and G are *equivalent* if F entails G and G entails F

13

- Satisfaction, entailment, and equivalence are *semantic* concepts – defined in terms of the actual relations in the "real world."
  - They define *what these notions are*, **not** how to compute them
- How to check if F entails f or if F and G are equivalent?
- Apply the respective definitions for all possible relations?
- *Bad idea*: might be infinite in number for infinite domains
- Even for finite domains, we have to look at relations of *all* arities
- **Solution**: find algorithmic, *syntactic* ways to compute these notions
- *Important*: The syntactic solution must be "correct" with respect to the semantic definitions
- Correctness has two aspects: *soundness* and *completeness*

14

Armstong's Axioms for FDs

- This is the *syntactic* way of computing/testing the various properties of FDs

- **Reflexivity**: If $Y \subseteq X$ then $X \rightarrow Y$ (trivial FD)
- *Name, Address* $\rightarrow$ *Name*

- **Augmentation**: If $X \rightarrow Y$ then $XZ \rightarrow YZ$
- If *Town* $\rightarrow$ *Zip* then *Town, Name* $\rightarrow$ *Zip, Name*

- **Transitivity**: If $X \rightarrow Y$ and $Y \rightarrow Z$ then $X \rightarrow Z$

15

More rules

- Two more rules (which can be derived from the axioms) can be useful:

- **Union**: If $X \rightarrow Y$ and $X \rightarrow Z$ then $X \rightarrow YZ$
- **Decomposition**: If $X \rightarrow YZ$ then $X \rightarrow Y$ and $X \rightarrow Z$

- And of course **transitivity**

16

Soundness and Completeness

- Axioms are *sound*: If an FD *f: X→ Y* can be derived from a set of FDs *F* using the axioms, then *f* holds in every relation that satisfies every FD in *F*.
- Axioms are *complete:* If *F* entails *f* , then *f* can be derived from *F* using the axioms
- A consequence of completeness is the following (naïve) algorithm to determining if *F* entails *f*:
- Algorithm: Use the axioms in all possible ways to generate *F+* (the set of possible FD's is finite so this can be done) and see if *f* is in *F+*

17

Reflexivity

- If Y $\subseteq$ X, then X -> Y
- Let R=(A,B,C,D,E), X = {ABCD}, Y={CD}
- e.g.:
- t1 =(a1,b1,c1,d1,e1)
- t2 =(a2,b2,c2,d2,e2)
- π_X(t1)= π_X(t2) a1 = a2,b1 = b2, ***c1 = c2,d1 = d2***
- => π_Y(t1)= π_Y(t2)

18

Augmentation

- If X -> Y, then XZ -> YZ for any Z

- Let R=(A,B,C,D,E), X={AB} Y={CD} Z={E}
- e.g.:
- t1 =(a1,b1,c1,d1,e1)
- t2 =(a2,b2,c2,d2,e2)
- $\pi_{XZ}(t1) = \pi_{XZ}(t2)$ => a1 = a2, b1 = b2, e1 = e2
- Since X ->Y and e1 = e2
- then c1 = c2, d1 = d2, e1 = e2
- => $\pi_{YZ}(t1) = \pi_{YZ}(t2)$

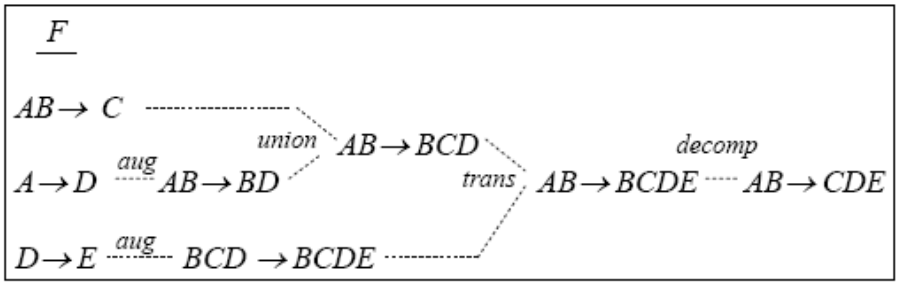
19

Transitivity

- If X -> Y, and Y -> Z then X -> Z
- Let R = (A,B,C,D,E), X={AB}, Y={CD} X={E}
- e.g.:
- t1 =(a1,b1,c1,d1,e1)
- t2 =(a2,b2,c2,d2,e2)
- assume X ->Y and Y -> Z
- $\pi_X(t1) = \pi_X(t2)$ => a1 = a2,b1 = b2
- Since X -> Y then c1 = c2 ,d1 = d2
- => $\pi_Y(t1) = \pi_Y(t2)$
- Since Y -> Y then e1 = e2
- => $\pi_Z(t1) = \pi_Z(t2)$

20

Generating F+



- Thus, $AB \rightarrow BD$, $AB \rightarrow BCD$, $AB \rightarrow BCDE$, and $AB \rightarrow CDE$ are all elements of $F+$

21

Attribute Closure

- Calculating *attribute closure* leads to a more efficient way of checking entailment
- The *attribute closure* of a set of attributes, X, with respect to a set of functional dependencies, F, (denoted $X+_F$) is the set of all attributes, A, such that $(X \rightarrow A) \in$ F+, i.e., $X+_F = \{A | (X \rightarrow A) \in F+\}$
- $X+_{F1}$ is not necessarily the same as $X+_{F2}$ if $F1 \neq F2$
- *Attribute closure and entailment*:
- Algorithm: Given a set of FDs, F, then $(X \rightarrow Y) \in F+$ if and only if $X+_F \supseteq Y$

22

Example

- *F: AB →C, A → D, D → E, AC → B*

X	$X+_F$
A	*{A, D, E}*
AB	*{A, B, C, D, E}*
(Hence *AB* is a key)	
B	*{B}*
D	*{D, E}*

- Is *AB → E* entailed by *F*? *Yes*
- Is *D→ C* entailed by *F*? *No*
- => $X+_F$ allows us to determine FDs of the form *X → Y* entailed by *F*

23

Computation of X_F+

closure := X; // since X ⊆ $X+_F$

repeat

old := closure;

if there is an FD *Z → V* in *F* such that *Z ⊆ closure*
and *V ⊆ closure*

then *closure := closure ∪ V*

until *old = closure*

- If *T ⊆ closure* then *X → T* is entailed by *F*

24

Example

- **Problem**: Compute the attribute closure of *AB* with respect to the set of FDs *:*
  - $AB \rightarrow C$ ---- (a)
  - $A \rightarrow D$ ----- (b)
  - $D \rightarrow E$ ------ (c)
  - $AC \rightarrow B$ ------ (d)

- Initially *closure = {AB}*
- Using (a) *closure = {ABC}*
- Using (b) *closure = {ABCD}*
- Using (c) *closure = {ABCDE}*

25

Normalization

- Each normal form is a set of conditions on a schema that guarantees certain properties (relating to redundancy and update anomalies)
- First normal form (1NF) is the same as the definition of relational model (relations = sets of tuples; each tuple = sequence of *atomic values*)
- Second normal form (2NF): *no non-key attribute is dependent on part of a key*; has no practical or theoretical value – won't discuss in detail
- The two commonly used normal forms are *third normal form* (3NF) and *Boyce-Codd normal form* (BCNF)

26

BCNF

- **Definition**: A relation schema **R** is in BCNF if for every FD $X \rightarrow Y$ associated with **R** either

1. $Y \subseteq X$ (i.e., the FD is trivial) *or*
2. X is a superkey of **R**

- *Example*: Person1(*PAN, Name, Address*)
- The only FD is *PAN* → *Name, Address*
- Since *PAN* is a key, Person1 is in BCNF

27

More examples

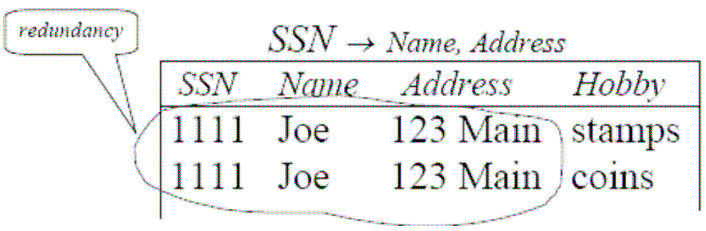
- Person (*PAN, Name, Address, Hobby*)
- The FD *PAN* → *Name, Address* does not satisfy requirements of BCNF since the key is (*PAN, Hobby*)

- HasAccount (*AccountNumber, ClientId, OfficeId*)
- The FD *AcctNum*→ *OfficeId* does not satisfy BCNF
- Requirements since keys are (*ClientId, OfficeId*) and (*AcctNum, ClientId*)

28

BCNF and Redundancy

- Suppose **R** has a FD $A \rightarrow B$. If an instance has 2 rows with same value in *A*, they *must* also have same value in *B* (=> redundancy, if the A-*value* repeats twice)
- If *A* is a superkey, there cannot be two rows with same value of *A* – Hence, BCNF eliminates redundancy



29

Decompositions

- **Goal**: Eliminate redundancy by decomposing a relation into several relations in a higher normal form
- Decomposition must be *lossless*: it must be possible to reconstruct the original relation from the relations in the decomposition
  - We will see why

30

Decompositions

- Schema **R** = (*R, F*)
  - *R* is set a of attributes
  - *F* is a set of functional dependencies over *R*
- The *decomposition of schema* **R** is a collection of schemas **R**i = (*Ri, Fi*) where
  - *R* = ∪ *i Ri* (*no new attributes*)
  - *Fi* is a set of functional dependences involving only attributes of *Ri*
  - *F* entails *Fi* for all *i* (*no new FDs*)
- The *decomposition of an instance*, **r**, of **R** is a set of relations **r***i* = π_{Ri}(**r**) for all *i*

31

Example

- Schema (*R, F*) where
  - *R* = {*PAN, Name, Address, Hobby*}
  - *F* = {*PAN*→ *Name, Address*}
- can be decomposed into
- *R1* = {*PAN, Name, Address*}
- *F1* = {*PAN* → *Name, Address*}
- and
- *R2* = {*PAN, Hobby*}
- *F2* = { }

32

Decomposition into BCNF

- Consider relation R with FDs F.
- If X -> Y violates BCNF (and X ∩ Y = ∅), decompose R into XY and R - Y.
- Repeated application of this idea will give us a collection of relations that are in BCNF 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.

33

Lossless Decomposition

- A decomposition should not lose information
- A decomposition (**R***1*,…,**R***n*) of a schema, **R**, is *lossless* if every valid instance, **r**, of **R** can be reconstructed from its components:

r = **r***1* ∞ **r***2* …… ∞ **r***n*

- where each **r**i = $\pi_{\mathbf{R}i}(\mathbf{r})$

34

Lossy Decomposition

- *The following is always the case* (Think why?):

$\mathbf{r} \subseteq \mathbf{r}1 \propto \mathbf{r}2 \ldots \propto \mathbf{r}n$

- *But the following is not always true*:

$\mathbf{r} \supseteq \mathbf{r}1 \propto \mathbf{r}2 \ldots \propto \mathbf{r}n$

Example: $\mathbf{r}$ $\not\supseteq$ $\mathbf{r}_1$ $\bowtie$ $\mathbf{r}_2$

SSN	Name	Address
1111	Joe	1 Pine
2222	Alice	2 Oak
3333	Alice	3 Pine

SSN	Name
1111	Joe
2222	Alice
3333	Alice

Name	Address
Joe	1 Pine
Alice	2 Oak
Alice	3 Pine

The tuples (2222, Alice, 3 Pine) and (3333, Alice, 2 Oak) *are in the join, but not in the original*

35

Why do we term it Lossy?

- In the previous example, the tuples (2222, Alice, 3 Pine) and (3333, Alice, 2 Oak) were *gained*, not lost!
- Why do we say that the decomposition was lossy?

- What was lost is ***information***:
- That 2222 lives at 2 Oak: *In the decomposition, 2222 can live at either 2 Oak or 3 Pine*
- That 3333 lives at 3 Pine: *In the decomposition, 3333 can live at either 2 Oak or 3 Pine*

36

Testing for LosslePANess

- A (binary) decomposition of **R** = (*R, F*) into **R**1 = (*R1, F1*) and **R**2 = (*R2, F2*) is lossless *if and only if* :
- either the FD $(R1 \cap R2) \rightarrow R1$ is in $F+$
- or the FD $(R1 \cap R2) \rightarrow R2$ is in $F+$

- Intuitively: the attributes common to R1 and R2 must contain a key for either R1 or R2.

37

Example

- Schema (*R, F*) where $R = \{PAN, Name, Address, Hobby\}$, $F = \{PAN \rightarrow Name, Address\}$ can be decomposed into

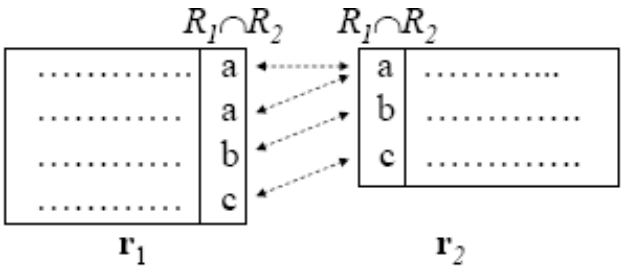
- $R1 = \{PAN, Name, Address\}$, $F1 = \{PAN \rightarrow Name, Address\}$ &
- $R2 = \{PAN, Hobby\}$, $F2 = \{\ \}$

- Since $R1 \cap R2 = PAN$ and $PAN \rightarrow R1$ the decomposition is lossless

38

The intuition behind this

- Suppose $R1 \cap R2 \rightarrow R2$. Then a row of **r***1* can combine with exactly one row of **r***2* in the natural join (since in **r***2* a particular set of values for the attributes in $R1 \cap R2$ defines a unique row)



39

Dependency Preservation

- Consider a decomposition of **R** = (*R, F*) into **R**1 = (*R1, F1*) and **R**2 = *(R2, F2)*
- An FD $X \rightarrow Y$ of *F* is in *Fi* iff $X \cup Y \subseteq Ri$
- An FD, $f \in F$ may be in neither *F1*, nor *F2,* nor even $(F1 \cup F2)+$
- Checking that *f* is true in **r**1 or **r**2 is (relatively) easy
- Checking *f* in $\mathbf{r}1 \propto \mathbf{r}2$ is harder – requires a join
- *Ideally*: want to check FDs **locally**, in **r**1 and **r**2, and have a guarantee that every $f \in F$ holds in $\mathbf{r}1 \propto \mathbf{r}2$
- The decomposition is *dependency preserving* iff the sets *F* and $F1 \cup F2$ are equivalent: $F+ = (F1 \cup F2)+$
- Then checking all FDs in *F,* as **r***1* and **r***2* are updated, can be done by checking *F1* in **r***1* and *F2* in **r***2*

40

- If *f* is an FD in *F,* but *f* is not in *F1* ∪ *F2,* there are two possibilities:

1. $f \in (F1 \cup F2)+$

- If the constraints in *F1* and *F2* are maintained, *f* will be maintained automatically.

2. $f \notin (F1 \cup F2)+$

- *f* can be checked only by first taking the join of r*1* and r*2*. This is costly.

41

Example

- Schema (*R, F*) where *R* = {*SSN, Name, Address, Hobby*} & *F* = {*SSN* → *Name, Address*} can be decomposed into

- *R1* = {*SSN, Name, Address*}, *F1* = {*SSN* → *Name, Address*} &
- *R2* = {*SSN, Hobby*}, *F2* = { }

- Since *F* = *F1* ∪ *F2* , it trivially holds that *F*+ = (*F1* ∪ *F2*)+ i.e., the decomposition is dependency preserving

42

Example

- Schema: (*ABC; F*) , *F* = {*A -> B, B-> C, C->B*}
- Decomposition:

1. (*AC*, *F*1), *F*1 = {*A->C*}

- Note: A->C ∉ *F,* but in F+

2. (*BC, F*2), *F*2 = {*B-> C, C-> B*}

- *A -> B* ∉ (*F*1 ∪ *F*2), **but** *A -> B* ∈ (*F*1 ∪ *F*2)+.

- So *F*+ = (*F*1 ∪ *F*2)+ and thus the decompositions is still dependency preserving

43

Example

- HasAccount (*AccountNumber, ClientId, OfficeId*)
  - *f*1: *AccountNumber* → *OfficeId*
  - *f*2: *ClientId, OfficeId* → *AccountNumber*
- Decomposition:
  - AcctOffice = (*AccountNumber, OfficeId*; {*AccountNumber* → *OfficeId*})
  - AcctClient = (*AccountNumber*, *ClientId*; {})
- Decomposition is lossless: *R1* ∩ *R2*= {*AccountNumber*} & *AccountNumber* → *OfficeId*
- In BCNF
- Not dependency preserving: *f2* ∉ (*F1* ∪ *F2*)+
- HasAccount *does not* have BCNF decompositions that are both lossless and dependency preserving! (Check, eg, by enumeration)
- Hence: BCNF+lossless+dependency preserving decompositions are not always possible!

44

BCNF Decomposition Algo

***Input*: R** = (*R; F*)

Decomp := **R**

while there is **S** = (*S; F'*) ∈ *Decomp* and **S** not in BCNF **do**

Find $X \rightarrow Y \in F'$ that violates BCNF // *X isn't a superkey in* **S**

Replace **S** in *Decomp* with **S1** = (*XY; F1*), **S2** = (*S* - (*Y* - *X*)*; F2*)

// *F1 = all FDs of F' involving only attributes of XY*

// *F2 = all FDs of F' involving only attributes of S - (Y - X)*

end

return *Decomp*

45

Example

- **Given: R** = (*R; T*) where *R = ABCDEFGH* and *T = {ABH→ C, A→ DE, BGH→ F, F→ ADH, BH→ GE}*
- **step 1:** Find a FD that violates BCNF

  Not *ABH → C* since *(ABH)+* includes all attributes (*BH* is a key)

  A → DE violates BCNF since *A* is not a superkey (*A+ =ADE*)
- **step 2:** Split **R** into:

  R1 = (*ADE, {A→ DE }*)

  R2 = (*ABCFGH; {ABH→ C, BGH→ F, F→ AH , BH→ G}*)

  Note 1: **R1** is in BCNF

  Note 2: Decomposition is *lossless* since *A* is a key of **R1**.

  Note 3: FDs *F → D* and *BH → E* are not in *T1* or *T2*. But both can be derived from *T1 ∪ T2*

  (*E.g., F→ A* and *A→ D* implies *F→ D)*

  Hence, decomposition is *dependency preserving*.

46

Properties of algorithm

- Let $X \rightarrow Y$ violate BCNF in **R** = (*R,F*) and **R1** = (*R1,F1*), **R2** = (*R2,F2*) is the resulting decomposition. Then:
- There are *fewer violations* of BCNF in **R1** and **R2** than there were in **R**
- $X \rightarrow Y$ implies *X* is a key of **R1**
- Hence $X \rightarrow Y \in F1$ does not violate BCNF in **R1** and, since
- $X \rightarrow Y \notin F2$, does not violate BCNF in **R2** either
- Suppose *f* is $X' \rightarrow Y'$ and $f \in F$ doesn't violate BCNF in **R**.
- If $f \in F1$ or *F2* it does not violate BCNF in **R1** or **R2** either since *X*' is a superkey of **R** and hence also of **R1** and **R2** .
- The decomposition is *lossless s*ince $F1 \cap F2 = X$

47

Example (contd)

- **Given: R2 =** (*ABCFGH;* {$ABH \rightarrow C$, $BGH \rightarrow F$, $F \rightarrow AH$, $BH \rightarrow G$})
- **step 1:** Find a FD that violates BCNF.
  Not $ABH \rightarrow C$ or $BGH \rightarrow F$, since *BH* is a key of **R2**
  $F \rightarrow AH$ violates BCNF since *F* is not a superkey ($F+ = AH$)
- **step 2:** Split **R2** into:
  R21 = (*FAH, {*$F \rightarrow AH$*}*)
  R22 = (*BCFG;* {})

  Note 1: Both **R21** and **R22** are in BCNF.

  Note 2: The decomposition is *lossless* (since *F* is a key of **R21**)

  Note 3: FDs $ABH \rightarrow C$, $BGH \rightarrow F$, $BH \rightarrow G$ are not in *T21* or *T22* , and they can't be derived from $T1 \cup T21 \cup T22$.
- Hence the decomposition is <u>*not* dependency-preserving</u>

48

More properties

- A BCNF decomposition is *not necessarily* dependency preserving
- But *always* lossless
- BCNF+lossless+dependency preserving is sometimes unachievable (recall HasAccount)

49

Third NF

- A relational schema **R** is in 3NF if for every FD $X \rightarrow Y$ associated with **R** *either*
  1. $Y \subseteq X$ (i.e., the FD is trivial) OR
  2. X is a superkey of **R** OR
  3. Every $A \in Y$ is part of some key of **R**

BCNF Cond.

- 3NF is weaker than BCNF (every schema in BCNF is also in 3NF)

50

Example

- HasAccount (*AcctNum, ClientId, OfficeId*)
- *ClientId*, *OfficeId* → *AcctNum*

- OK since LHS contains a key
- *AcctNum* → *OfficeId*
- OK since RHS is part of a key

- HasAccount is in 3NF but it might still contain redundant information due to *AcctNum* → *OfficeId* (which is not allowed by BCNF)

51

3NF Example

HasAccount might store redundant data:

ClientId	*OfficeId*	*AcctNum*
1111	Stony Brook	28315
2222	Stony Brook	28315
3333	Stony Brook	28315

3NF: *OfficeId* part of key
FD: *AcctNum* → *OfficeId*

redundancy

- Decompose to eliminate redundancy:

ClientId	*AcctNum*
1111	28315
2222	28315
3333	28315

BCNF (only trivial FDs)

OfficeId	*AcctNum*
Stony Brook	28315

BCNF: *AcctNum* is key
FD: *AcctNum* → *OfficeId*

Another Example

- Person (*SSN, Name, Address, Hobby*)
- (*SSN, Hobby*) is the only key.
- *SSN*→ *Name* violates 3NF conditions since *Name* is not part of a key and *SSN* is not a superkey

53

Third Normal Form

- Compromise – Not all redundancy removed, but dependency preserving decompositions are always possible (and, of course, lossless)
- 3NF decomposition is based on a *minimal cover*

54

Minimal Cover

- A *minimal cover* of a set of dependencies, *T,* is a set of dependencies, *U,* such that:

1. *U* is equivalent to *T* (*T*+ = *U*+)
2. All FDs in *U* have the form $X \rightarrow A$ where *A* is a single attribute
3. It is not possible to make *U* smaller (while preserving equivalence) by
   - Deleting an FD OR
   - Deleting an attribute from an FD (either from LHS or RHS)

- FDs and attributes that can be deleted in this way are called *redundant*

55

Finding the minimal cover

- **Example**:

R = (R, T), R = (A, B, C, D, E, F, G, H, K)

$T = \{ABH \rightarrow CK, A \rightarrow D, C \rightarrow E, BGH \rightarrow F, F \rightarrow AD, E \rightarrow F, BH \rightarrow E\}$

- **Step 1**: Make RHS of each FD into a single attribute

  Algorithm: Use the decomposition inference rule for FDs

  Example: $F \rightarrow AD$ replaced by $F \rightarrow A$, $F \rightarrow D$; $ABH \rightarrow CK$ by $ABH \rightarrow C$, $ABH \rightarrow K$

56

- **step 2**: Eliminate redundant attributes from LHS.

  Algorithm: If FD $XB \rightarrow A \in T$ (where B is a single attribute) and $X \rightarrow A$ is entailed by T, then B was unnecessary
- Example: Can an attribute be deleted from $ABH \rightarrow C$?
- Compute $AB+_T$, $AH+_T$, $BH+_T$.
- Since $C \in (BH)+_T$, $BH \rightarrow C$ is entailed by T and A is redundant in $ABH \rightarrow C$.

57

- **step 3**: Delete redundant FDs from T

  Algorithm: If $T - \{f\}$ entails f, then f is redundant

  If f is $X \rightarrow A$ then check if $A \in X+_T-\{f\}$

  Example: $BH \rightarrow F$ is entailed by $E \rightarrow F$, $BH \rightarrow E$, so it is redundant
- *Note*: Steps 2 and 3 cannot be reversed!!

58

Synthesizing 3NF Schemas

Starting with a schema **R** = (*R, T*)

step 1: Compute a minimal cover, *U*, of *T*. The decomposition is based on *U*, but since *U*+ = *T*+ the same functional dependencies will hold

A minimal cover for *T*={*ABH*→*CK, A*→*D, C*→*E, BGH*→*F, F*→*AD, E*→ *F, BH* → *E*}

is

U={*BH*→*C, BH*→*K, A*→*D, C*→*E, F*→*A, E*→*F*}

59

step 2: Partition *U* into sets *U1, U2, … Un* such that the LHS of all elements of *Ui* are the same

U1 = {*BH* → *C, BH* → *K*},

U2 = {*A* → *D*},

U3 = {*C* → *E*},

U4 = {*F* → *A*},

U5 = {*E* → *F*}

60

step 3: For each *Ui* form schema **Ri** = (*Ri, Ui*), where *Ri* is the set of all attributes mentioned in *Ui*

Each FD of *U* will be in some **Ri.** Hence the decomposition is *dependency preserving*

R1 = (*BHCK; BH* → *C, BH* → *K*),

R2 = (*AD; A* → *D*),

R3 = (*CE; C* → *E*),

R4 = (*FA; F* → *A*),

R5 = (*EF; E* → *F*)

61

step 4: If no *Ri* is a superkey of **R,** add schema **R0** = (*R0,*{ }) where *R0* is a key of **R**.

R0 = (*BGH,* { })

R0 might be needed when not all attributes are necessarily contained in *R*1 ∪ *R*2 … ∪ *R*n

A missing attribute, *A*, must be part of all keys (since it's not in any FD of *U*, deriving a key constraint from *U* involves the augmentation axiom)

R0 might be needed even if all attributes are accounted for in *R*1 ∪ *R*2 … ∪ *R*n

Example: (*ABCD*; {*A->B*, *C->D*}). Step 3 decomposition: *R*1 = (*AB*; {*A->B*}), *R*2 = (*CD*; {*C->D*}). Lossy! Need to add (AC; { }), for losslessness

- Step 4 guarantees lossless decomposition.

62

BCNF Design Strategy

- The resulting decomposition, **R0, R1, … Rn** , is

1. Dependency preserving (since every FD in *U* is a FD of some schema)
2. Lossless (although this is not obvious)
3. In 3NF (although this is not obvious)

- Strategy for decomposing a relation

1. Use 3NF decomposition first to get lossless, dependency preserving decomposition
2. If any resulting schema is not in BCNF, split it using the BCNF algorithm (but this may yield a non-dependency preserving result)

63

Limitations

- By limiting redundancy, normalization helps maintain consistency and saves space
- But performance of querying can suffer because related information that was stored in a single relation is now distributed among several

```
Student = (Id, Name, Address, Status)
Transcript = (StudId, CrsCode, SectionNo,
  Semester, Year, Grade)
```

- **Example**: A join is required to get the names and grades of all students taking CS317 in F2007.

```
SELECT S.Name, T.Grade
FROM Student S, Transcript T
WHERE S.Id = T.StudId AND
T.CrsCode = 'CS317' AND T.Semester = 'F2007'
```

64

Denormalization

- **Tradeoff**: *Judiciously* introduce redundancy to improve performance of certain queries
- **Example**: Add attribute *Name* to Transcript (making it Transcript')

```
SELECT T.Name, T.Grade
FROM Transcript' T
WHERE T.CrsCode = 'CS317' AND T.Semester = 'F2007'
```

- Join is avoided
- If queries are asked more frequently than Transcript is modified, added redundancy might improve average performance
- But, Transcript' is no longer in BCNF since key is (*StudId, CrsCode, Semester*) and *StudId* → *Name*

65

Attribute Independence in BCNF

- BCNF schemas can have redundancy, e.g., when we force two or more many-many relationships in a single relation.
- Consider the schema Courses(Number, DeptName, Textbook, Instructor).
  - Each Course can have multiple required Textbooks.
  - Each Course can have multiple Instructors.
  - Instructors can use any required textbook in a Course.

Number	DeptName	Textbook	Instructor
4604	CS	FCDB	Murali
4604	CS	SQL Made Easy	Murali
4604	CS	FCDB	Ramakrishan
4604	CS	SQL Made Easy	Ramakrishan

66

- Is the relation in BCNF?
- Yes, there are NO non trivial FDs.

- Is there redundancy?
- Yes – in the TextBook and Instructor attributes.
- We can remove the redundancy by decomposing Courses into

Courses1(Number, DeptName, Textbook)

Courses2(Number, DeptName, Instructor)

67

- FDs and BCNF are not rich enough to express these types of redundancies.

68

MVD

- A multi-valued dependency (MVD or MD) is an assertion that two sets of attributes are independent of each other.
- The multi-valued dependency A1 A2 . . .An ->> B1 B2 . . .Bm holds in a relation R if for every pair of tuples t and u in R that agree on all the A's, we can find a tuple v in R that agrees

1. with both t and u on A's,
2. with t on the B's, and
3. with u on all those attributes of R that are not A's or B's.

69

Number	DeptName	Textbook	Instructor
4604	CS	FCDB	Murali
4604	CS	SQL Made Easy	Murali
4604	CS	FCDB	Ramakrishan
4604	CS	SQL Made Easy	Ramakrishan

70

Example

- **`Number DeptName ->>Textbook`** is an MD. For every pair of tuples t and u that agree on Number and DeptName, we can find a tuple v that agrees

1. with both t and u on Number and DeptName,
2. with t on Textbook, and with u on Instructor.

- **`Number DeptName ->> Instructor`** is an MD. For every pair of tuples t and u that agree on Number and DeptName, we can find a tuple v that agrees

1. with both t and u on Number and DeptName,
2. with t on Instructor, and with u on Textbook.

71

Facts about MDs

- Given tuples t, u, and v that satisfy an MD, we can infer the existence of another tuple w that agrees

1. with both t and u on A's,
2. with u on the B's, and
3. with t on all those attributes of R that are not A's or B's.

- Every FD A -> B is an MD A ->> B.
- Proof: make u and v the same tuple.
- Definition of keys depends on FDs and not on MDs.

72

MD Manipulation Rules

- Trivial dependencies rule: If A->> B is an MD, then A ->> AB is also an MD.
- Transitive rule: if A ->> B and B ->> C are MDs, then A ->> C is an MD.
- Complementation rule: if A ->> B, then A ->> C is an MD, where C is the set of all attributes not in the MD.
- The splitting rule does not hold! If A ->> BC is an MD, then it is not true that A ->> B and A ->> C are MDs.

73

Triviality

- An MD A1 A2 . . .An ->> B1 B2 . . .Bm for a relation R is non-trivial if

1. **{A1,A2,...,An}∩{B1,B2,...,Bm} = Φ;** i.e., none of the B's is an A AND

2. **{A1,A2,...,An} ∪{B1,B2,...,Bm}** is not the set of all the attributes of R.

74

Fourth Normal Form

- A relation R is in fourth normal form (4NF) if for every non-trivial MD A1 A2 . . .An ->> B1B2 . . .Bm, {A1,A2, . . . ,An} is a superkey.
- A relation in 4NF is also in BCNF since an FD is a special case of an MD.

75

Decomposition into 4NF

- Suppose R is a relation with a set X of attributes and A1A2 . . .An ->> B1B2 . . .Bm violates 4NF.

1. Decompose R into two relations whose attributes are the As and the Bs, i.e., {A1,A2, . . . ,An,B1,B2, . . . ,Bm} and
2. all the attributes of R that are not B's, i.e., X − {B1,B2, . . . ,Bm}.
3. Check if the new relations are in 4NF.

- Projecting MDs: out of the scope of 317. In practice, we can use rules for manipulating MDs to project them.
- Date-Fagin theorem: if a relation schema is in BCNF and has a key with one attribute, then it is in 4NF.

76

Relationships amongst Normal Forms

- 4NF implies BCNF, i.e., if a relation is in 4NF, it is also in BCNF.
- BCNF implies 3NF, i.e., if a relation is in BCNF, it is also in 3NF.

Property	3NF	BCNF	4NF
Eliminates redundancy due to FDs	Maybe	Yes	Yes
Eliminates redundancy due to MDs	No	No	Yes
Preserves FDs	Yes	Maybe	Maybe
Preserves MDs	Maybe	Maybe	Maybe

77

Kth NF

- First Normal Form: each attribute is atomic.
- Second Normal Form: No non-trivial FD has a left side that is a proper subset of a key.
- Third Normal Form: we just discussed it.
- Fourth Normal Form: we just discussed it.
- Fifth Normal Form: outside the scope of 317.
- Sixth Normal Form: different versions exist. One version is newly developed for temporal databases.
- Seventh Normal Form: your ticket to fame and fortune.

78