Relational Normalization Theory

- Database Constraints
- Database Design

Redundancy

- Dependencies between attributes cause redundancy
  - Ex. All the people working in the same place have the same zip code

SSN	*Name*	*Department*	*Zip*
1234	Joe	DCS	M5S 3H5
4321	Mary	DCS	M5S 3H5
•	Tom	DCS	M5S 3H5
.....			

Redundancy

SSN	*Name*	*Address*	*Hobby*
123456789	Mike	123 College St.	stamps
123456789	Mike	123 College St.	coins
.....			

Redundancy

Redundancy and Other Problems

- Example: Person (*SSN, 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
  - *SSN* is the key of the entity set, but (*SSN, Hobby*) is the key of the corresponding relation
    - The relation Person can't describe people without hobbies

Anomalies

- 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

Decomposition

- **Solution**: use two relations to store Person information
  - Person1 (*SSN, Name, Address*)
  - Hobbies (*SSN, 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
  - We can represent persons who do not have hobbies

Normalization Theory

- Result of E-R analysis needs 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*)

Functional Dependencies

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

- **Definition**: An FD $\boldsymbol{X} \rightarrow \boldsymbol{Y}$ 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*

Key Constraint

- Is a special kind of functional dependency:
  - Let *K* be a set of attributes of R, and *U* the set of **all** attributes of R. Then *K* is a *key* if the functional dependency $K \rightarrow U$ is satisfied in R.
    - *SSN* $\rightarrow$ *SSN, Name, Address* (in the Person1 relation)
- A *candidate key* is a minimal superkey
  - *K* is a key in *R*, if for each X $\subset$ K, *X* is not a key
    - *SSN, Hobby* $\rightarrow$ *SSN, Name, Address, Hobby* but
    - *SSN* $\rightarrow$ *SSN, Name, Address, Hobby*
    - *Hobby* $\rightarrow$ *SSN, Name, Address, Hobby*
- A *prime attribute* is an attribute of a key

Functional Dependencies cont'd

- *Address* $\rightarrow$ *ZipCode*
  - DCS's ZIP is M5S 3H5
- *Author, Title, Edition* $\rightarrow$ *PublicationDate*
  - Database Management Systems, by R. Ramakrishnan, and J. Gehrke, McGraw Hill, 2003 (3rd Edition)
- *CourseID* $\rightarrow$ *ExamDate, ExamTime*
  - CSC343's exam date is August 9, starting at 6pm

Entailment, Closure, 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 *Phone#* $\rightarrow$ *Address, Address* $\rightarrow$ *ZipCode* then

      Phone# $\rightarrow$ *ZipCode*
- **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***

Armstrong'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 *Address* $\rightarrow$ *ZipCode* then *Address, Name* $\rightarrow$ *ZipCode, Name*
- **Transitivity**: If $X \rightarrow Y$ and $Y \rightarrow Z$ then $X \rightarrow Z$
  - If *Phone#* $\rightarrow$ *Address* and *Address* $\rightarrow$ *ZipCode*, then *Phone#* $\rightarrow$ *ZipCode*

Soundness

- Axioms are *sound*: If an FD *f:* $X \rightarrow Y$ can be derived from a set of FDs ***F*** using the axioms, then *f* holds in every relation that satisfies every FD in ***F.***
- Example: Given $X \rightarrow Y$ and $X \rightarrow Z$ then

$X \rightarrow XY$ *Augmentation by X*
$YX \rightarrow YZ$ *Augmentation by Y*
$X \rightarrow YZ$ *Transitivity*

  - Thus, $X \rightarrow Y\,Z$ is satisfied in every relation where both $X \rightarrow Y$ and $X \rightarrow Z$ are satisfied
    - Therefore, we have derived the *union rule* for FDs

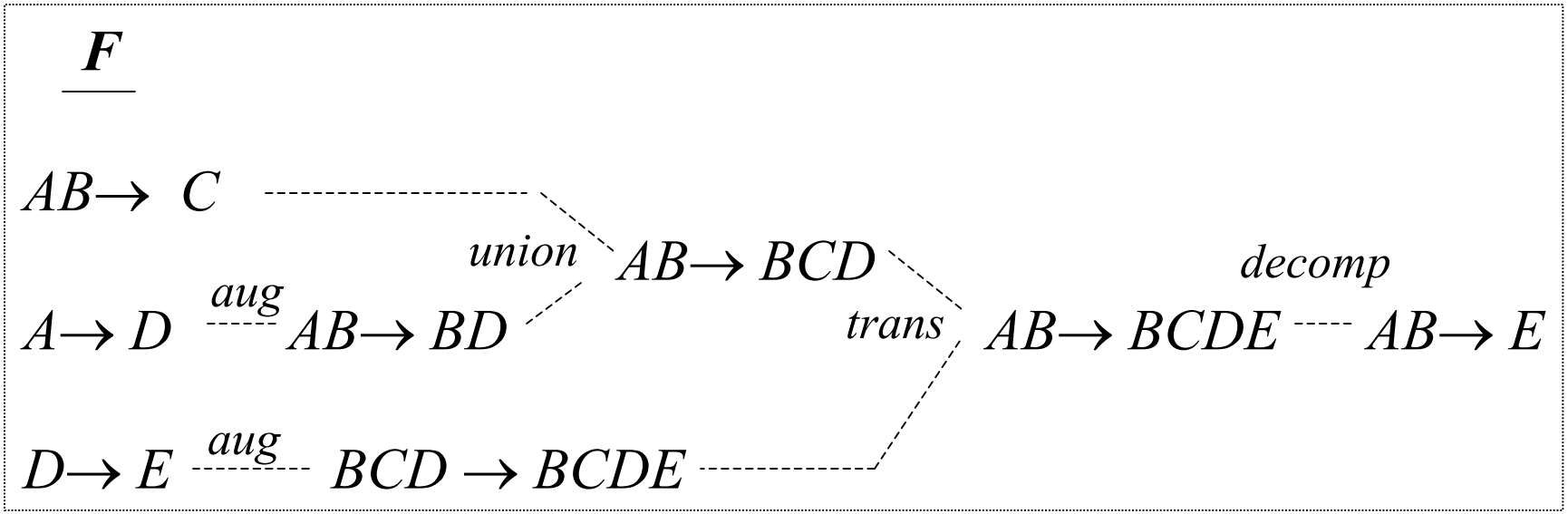
Completeness

- 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^+

Correctness

- The notions of *soundness* and *completeness* link the syntax (Armstrong's axioms) with semantics (the definitions in terms of relational instances)
- This is a precise way of saying that the algorithm for entailment based on the axioms is ``correct'' with respect to the definitions

Generating F^+



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

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, $\boldsymbol{F}$, (denoted X^{+}_{F}) is the set of all attributes, A, such that $X \rightarrow A$
  - X^{+}_{F} is not necessarily the same as X^{+}_{G} if $\boldsymbol{F} \neq \boldsymbol{G}$
- *Attribute closure and entailment*:
  - Algorithm: Given a set of FDs, $\boldsymbol{F}$, then $X \rightarrow Y$ if and only if $Y \subseteq X^{+}_{F}$

Example - Computing Attribute Closure

F: $AB \rightarrow C$
$A \rightarrow D$
$D \rightarrow E$
$AC \rightarrow B$

X	X_F^+
A	$\{A, D, E\}$
AB	$\{A, B, C, D, E\}$
AC	$\{A, C, B, D, E\}$
B	$\{B\}$
D	$\{D, E\}$

Is $AB \rightarrow E$ entailed by ***F***? *Yes*
Is $D \rightarrow C$ entailed by ***F***? *No*

Result: X_F^+ allows us to determine FDs of the form
$X \rightarrow Y$ entailed by ***F***

Computation of Attribute Closure X^+_F

closure := *X*; // since $X \subseteq X^+_F$
repeat
  old := *closure*;
  if there is an FD $Z \rightarrow V$ in ***F*** such that
    $Z \subseteq$ *closure* **and** $V \subseteq$ *closure*
  then *closure* := *closure* $\cup$ *V*
until *old* = *closure*

– If $T \subseteq$ *closure* then $X \rightarrow T$ is entailed by ***F***

Normal Forms

- Each normal form is a set of conditions on a schema that guarantees certain properties (relating to redundancy and update anomalies)

- The two commonly used normal forms are *third normal form* (3NF) and *Boyce-Codd normal form* (BCNF)

BCNF

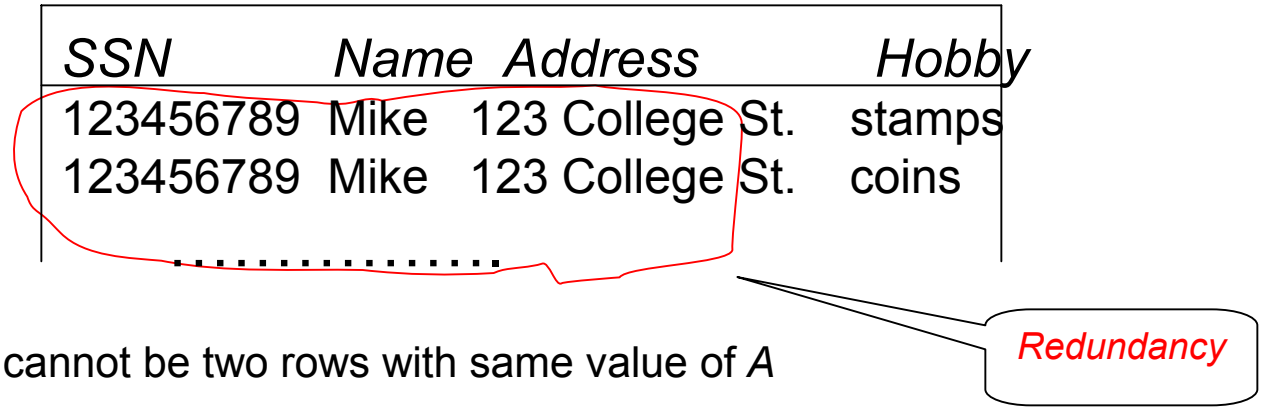
- **Definition**: A relation schema **R** is in BCNF if for every FD $X \rightarrow Y$ associated with **R** either
  - $Y \subseteq X$ (i.e., the FD is trivial) or
  - X is a key of **R**
- **Example**: Person1(*SSN, Name, Address*)
  - The only FD is *SSN* $\rightarrow$ *Name, Address*
  - Since *SSN* is a key, Person1 is in BCNF

(non) BCNF Examples

- Person (*SSN, Name, Address, Hobby*)
  - The FD *SSN* → *Name, Address* does not satisfy requirements of BCNF
    - since the key is (*SSN, Hobby*)
- HasAccount (*AcctNum, ClientId, OfficeId*)
  - The FD *AcctNum*→ *OfficeId* does not satisfy BCNF requirements
    - since keys are (*ClientId, OfficeId*) and (*AcctNum, ClientId*); not *AcctNum*.

Redundancy

- Suppose **R** has an FD $A \rightarrow B$, *and A is not a key.* 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 key, there cannot be two rows with same value of *A*
  - Hence, BCNF eliminates redundancy

Decomposition

- Schema **R** = (*R*, ***F***)
  - *R* is set a of attributes
  - ***F*** is a set of functional dependencies over *R*
    - Each key is described by a FD
- The *decomposition of schema* **R** is a collection of schemas

$\mathbf{R}_i = (R_i, \boldsymbol{F}_i)$ where

  - $R = \cup_i R_i$ for all *i* (*no new attributes*)
  - $\boldsymbol{F}_i$ is a set of functional dependences involving only attributes of R_i
  - ***F*** entails $\boldsymbol{F}_i$ for all *i* (*no new FDs*)
- The *decomposition of an instance*, **r**, of **R** is a set of relations

$\mathbf{r}_i = \pi_{Ri}(\mathbf{r})$ for all *i*

Example Decomposition

Schema (R, F) where

R = {*SSN, Name, Address, Hobby*}

$\boldsymbol{F}$ = {*SSN*$\rightarrow$ *Name, Address*}

can be decomposed into

R_1 = {*SSN, Name, Address*}

$\boldsymbol{F}_1$ = {*SSN* $\rightarrow$ *Name, Address*}

and

R_2 = {*SSN, Hobby*}

$\boldsymbol{F}_2$ = { }

BCNF Decomposition Algorithm

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

Decomp := **R**
while there is **S** = (*S; **F'***) $\in$ *Decomp* and **S** not in BCNF **do**
    Find $X \rightarrow Y \in$ ***F'*** that violates BCNF *// X isn't a key in* **S**
    Replace **S** in *Decomp* with
    $\mathbf{S_1} = (XY; \mathbf{F}_1)$, $\mathbf{S_2} = ((S - Y) \cup X); \mathbf{F}_2)$
    *// $\mathbf{F}_1$ = all FDs of **F'** involving only attributes of XY*
    *// $\mathbf{F}_2$ = all FDs of **F'** involving only attributes of $(S - Y) \cup X$*
end
return *Decomp*

Simple Example

- HasAccount :

(ClientId, OfficeId, AcctNum)

ClientId,*OfficeId* → *AcctNum*
AcctNum → *OfficeId*

- Decompose using *AcctNum* → *OfficeId* :

(OfficeId, AcctNum)

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

(ClientId , AcctNum)

BCNF (only trivial FDs)

A Larger Example

Given: **R** = (*R*; ***F***) where *R* = *ABCDEGHK* and
F = *{ABH→ C, A→ DE, BGH→ K, K→ ADH, BH→ GE}*

step 1: Find a FD that violates BCNF
Not $ABH \rightarrow C$ since $(ABH)^+$ includes all attributes
(*BH* is a key)
$A \rightarrow DE$ violates BCNF since *A* is not a key ($A^+ = ADE$)

step 2: Split **R** into:
$\mathbf{R_1}$ = (*ADE*, $\boldsymbol{F_1}$={*A→ DE* })
$\mathbf{R_2}$ = (*ABCGHK*; $\boldsymbol{F_1}$={*ABH→C, BGH→K, K→AH, BH→G*})
Note 1: $\mathbf{R_1}$ is in BCNF
Note 2: Decomposition is *lossless* since *A* is a key of $\mathbf{R_1}$.
Note 3: FDs $K \rightarrow D$ and $BH \rightarrow E$ are not in $\boldsymbol{F_1}$ or $\boldsymbol{F_2}$. But both can be derived from $\boldsymbol{F_1} \cup \boldsymbol{F_2}$
(*E.g., K→ A* and *A→ D* implies *K→ D)*
Hence, decomposition *is dependency preserving*.

Example cont'd

Given: $\mathbf{R_2}$ = (*ABCGHK; {ABH→C, BGH→K, K→AH, BH→G}*)

step 1: Find a FD that violates BCNF.

Not $ABH \rightarrow C$ or $BGH \rightarrow K$, since *BH* is a key of $\mathbf{R_2}$

$K \rightarrow AH$ violates BCNF since *K* is not a superkey ($K^+ = AH$)

step 2: Split $\mathbf{R_2}$ into:

$\mathbf{R_{21}}$ = (*KAH*, $\boldsymbol{F_{21}}$={$K \rightarrow AH$})

$\mathbf{R_{22}}$ = (*BCGK;* $\boldsymbol{F_{22}}$={})

Note 1: Both $\mathbf{R_{21}}$ and $\mathbf{R_{22}}$ are in BCNF.

Note 2: The decomposition is *lossless* (since *K* is a key of $\mathbf{R_{21}}$)

Note 3: FDs $ABH \rightarrow C$, $BGH \rightarrow K$, $BH \rightarrow G$ are not in $\boldsymbol{F_{21}}$ or $\boldsymbol{F_{22}}$, and they can't be derived from $\boldsymbol{F_1} \cup \boldsymbol{F_{21}} \cup \boldsymbol{F_{22}}$.

Hence the decomposition is *not* dependency-preserving

Lossless Schema Decomposition

- A decomposition should not lose information
- A decomposition ($\mathbf{R}_1,\ldots,\mathbf{R}_n$) of a schema, **R**, is *lossless* if every valid instance, **r**, of **R** can be reconstructed from its components:

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

$$\mathbf{r} = \mathbf{r}_1 \bowtie \mathbf{r}_2 \bowtie \ldots\ldots \bowtie \mathbf{r}_n$$

Lossy Decomposition

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

$$\mathbf{r} \subseteq \mathbf{r}_1 \bowtie \mathbf{r}_2 \bowtie \ldots \bowtie \mathbf{r}_n$$

But the following is not always true:

$$\mathbf{r} \supseteq \mathbf{r}_1 \bowtie \mathbf{r}_2 \bowtie \ldots \bowtie \mathbf{r}_n$$

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*

Lossy Decompositions: *What is Actually Lost?*

- 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*

Testing for Losslessness

- A (binary) decomposition of **R** = (*R*, ***F***) into $\mathbf{R}_1 = (R_1, \boldsymbol{F}_1)$ and $\mathbf{R}_2 = (R_2, \boldsymbol{F}_2)$ is lossless *if and only if* :
  - either the FD
    - $(R_1 \cap R_2) \rightarrow R_1$ is in $\boldsymbol{F}^+$
  - or the FD
    - $(R_1 \cap R_2) \rightarrow R_2$ is in $\boldsymbol{F}^+$

Example

Schema (R, F) where

R = {*SSN, Name, Address, Hobby*}

$\boldsymbol{F}$ = {*SSN* $\rightarrow$ *Name, Address*}

can be decomposed into

R_1 = {*SSN, Name, Address*}

$\boldsymbol{F}_1$ = {*SSN* $\rightarrow$ *Name, Address*}

and

R_2 = {*SSN, Hobby*}

$\boldsymbol{F}_2$ = { }

Since $R_1 \cap R_2 = SSN$ and $SSN \rightarrow R_1$ the decomposition is lossless

Dependency Preservation

- Consider a decomposition of **R** = (*R*, ***F***) into $\mathbf{R}_1 = (R_1, \boldsymbol{F}_1)$ and $\mathbf{R}_2 = (R_2, \boldsymbol{F}_2)$
  - An FD $X \rightarrow Y$ of $\boldsymbol{F}^+$ is in $\boldsymbol{F}_i$ iff $X \cup Y \subseteq R_i$
  - An FD, $f \in \boldsymbol{F}^+$ may be in neither $\boldsymbol{F}_1$, nor $\boldsymbol{F}_2$, nor even $(\boldsymbol{F}_1 \cup \boldsymbol{F}_2)^+$
    - Checking that *f* is true in $\mathbf{r}_1$ or $\mathbf{r}_2$ is (relatively) easy
    - Checking *f* in $\mathbf{r}_1$ $\mathbf{r}_2$ is harder – requires a join
    - *Ideally*: want to check FDs locally, in $\mathbf{r}_1$ and $\mathbf{r}_2$, and have a guarantee that every $f \in F$ holds in $\mathbf{r}_1 \bowtie \mathbf{r}_2$
- The decomposition is ***dependency preserving*** iff the sets ***F*** and $\boldsymbol{F}_1 \cup \boldsymbol{F}_2$ are equivalent: $\boldsymbol{F}^+ = (\boldsymbol{F}_1 \cup \boldsymbol{F}_2)^+$
  - Then checking all FDs in ***F***, as $\mathbf{r}_1$ and $\mathbf{r}_2$ are updated, can be done by checking $\boldsymbol{F}_1$ in $\mathbf{r}_1$ and F_2 in $\mathbf{r}_2$ $\bowtie$

Dependency Preservation

- If f is an FD in $\boldsymbol{F}$, but f is not in $\boldsymbol{F}_1 \cup \boldsymbol{F}_2$, there are two possibilities:
  - $f \in (\boldsymbol{F}_1 \cup \boldsymbol{F}_2)^+$
    - If the constraints in $\boldsymbol{F}_1$ and $\boldsymbol{F}_2$ are maintained, f will be maintained automatically.
  - $f \notin (\boldsymbol{F}_1 \cup \boldsymbol{F}_2)^+$
    - f can be checked only by first taking the join of $\mathbf{r}_1$ and $\mathbf{r}_2$. This is costly.

Example

Schema (R, $\boldsymbol{F}$) where

$R = \{SSN, Name, Address, Hobby\}$

$\boldsymbol{F} = \{SSN \rightarrow Name, Address\}$

can be decomposed into

$R_1 = \{SSN, Name, Address\}$

$\boldsymbol{F}_1 = \{SSN \rightarrow Name, Address\}$

and

$R_2 = \{SSN, Hobby\}$

$\boldsymbol{F}_2 = \{\ \}$

Since $\boldsymbol{F} = \boldsymbol{F}_1 \cup \boldsymbol{F}_2$ the decomposition is dependency preserving

Example

- Schema: (ABC; F) , $\boldsymbol{F} = \{A \rightarrow B, B \rightarrow C, C \rightarrow B\}$
- Decomposition:
  - (AC, $\boldsymbol{F}_1$), $\boldsymbol{F}_1 = \{A \rightarrow C\}$
    - Note: $A \rightarrow C \notin \boldsymbol{F}$, but in $\boldsymbol{F}^+$
  - (BC, $\boldsymbol{F}_2$), $\boldsymbol{F}_2 = \{B \rightarrow C, C \rightarrow B\}$

- $A \rightarrow B \notin (\boldsymbol{F}_1 \cup \boldsymbol{F}_2)$, **but** $A \rightarrow B \in (\boldsymbol{F}_1 \cup \boldsymbol{F}_2)^+$.
  - So $\boldsymbol{F}^+ = (\boldsymbol{F}_1 \cup \boldsymbol{F}_2)^+$ and thus the decompositions is still dependency preserving

Example

- HasAccount (*AcctNum, ClientId, OfficeId*)

  f_1: *AcctNum* $\rightarrow$ *OfficeId*

  f_2: *ClientId, OfficeId* $\rightarrow$ *AcctNum*

- Decomposition:

  R_1 = (*AcctNum, OfficeId*; {*AcctNum* $\rightarrow$ *OfficeId*})

  R_2 = (*AcctNum*, *ClientId*; {})

- Decomposition is lossless:

  $R_1 \cap R_2$= {*AcctNum*} and *AcctNum* $\rightarrow$ *OfficeId*

- In BCNF

- Not dependency preserving: $f_2 \notin (\boldsymbol{F}_1 \cup \boldsymbol{F}_2)^+$

- 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 achievable!

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*

Third Normal Form

- A relational schema **R** is in 3NF if for every FD $X \rightarrow Y$ associated with **R** either:
- $Y \subseteq X$ (i.e., the FD is trivial); or
  - X is a superkey of **R;** or
  - Every $A \in Y$ is part of some key of **R**
- 3NF is weaker than BCNF (every schema that is in BCNF is also in 3NF)

BCNF conditions

3NF 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)

3NF (Non) Example

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

Minimal Cover

- A *minimal cover* of a set of dependencies, ***F,*** is a set of dependencies, ***U,*** such that:
  - *U* is equivalent to ***F*** (***F***$^+$ = ***U***$^+$)
  - All FDs in ***U*** have the form $X \rightarrow A$ where *A* is a single attribute
  - It is not possible to make ***U*** smaller (while preserving equivalence) by
    - Deleting an FD
    - Deleting an attribute from an FD (either from LHS or RHS)
  - FDs and attributes that can be deleted in this way are called *redundant*

Computing Minimal Cover

- **Example**: $\boldsymbol{F}$ = {$ABH \rightarrow CK$, $A \rightarrow D$, $C \rightarrow E$,
  $BGH \rightarrow L$, $L \rightarrow AD$, $E \rightarrow L$, $BH \rightarrow E$}
- **step 1**: Make RHS of each FD into a single attribute
  - *Algorithm*: Use the decomposition inference rule for FDs
  - Example: $L \rightarrow AD$ replaced by $L \rightarrow A$, $L \rightarrow D$;
    $ABH \rightarrow CK$ by $ABH \rightarrow C$, $ABH \rightarrow K$
- **step 2**: Eliminate redundant attributes from LHS.
  - *Algorithm*: If FD $XB \rightarrow A \in \boldsymbol{F}$ (where B is a single attribute) and $X \rightarrow A$ is entailed by $\boldsymbol{F}$, then B was unnecessary
  - Example: Can an attribute be deleted from $ABH \rightarrow C$?
    - Compute AB^+_F, AH^+_F, BH^+_F.
    - Since $C \in (BH)^+_F$, $BH \rightarrow C$ is entailed by $\boldsymbol{F}$ and A is redundant in $ABH \rightarrow C$.

Computing Minimal Cover cont'd

- **step 3**: Delete redundant FDs from ***G***
  - *Algorithm*: If ***G*** – {*f*} entails *f*, then *f* is redundant
    - If *f* is $X \rightarrow A$ then check if A $\in X^{+}_{G-\{f\}}$
  - Example: $BH \rightarrow L$ is entailed by $E \rightarrow L$, $BH \rightarrow E$, so it is redundant

- *Note*: The order of steps 2 and 3 cannot be interchanged!!

Synthesizing a 3NF Schema

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

- **step 1**: Compute a minimal cover, ***U***, of ***F***. The decomposition is based on ***U***, but since $\boldsymbol{U}^+ = \boldsymbol{F}^+$ the same functional dependencies will hold

  A minimal cover for

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

  is

  $U = \{BH \rightarrow C,\ BH \rightarrow K,\ A \rightarrow D,\ C \rightarrow E,\ L \rightarrow A,\ E \rightarrow L\}$

Synthesizing a 3NF schema cont'd

- **step 2**: Partition $\boldsymbol{U}$ into sets $\boldsymbol{U}_1, \boldsymbol{U}_2, \ldots \boldsymbol{U}_n$ such that the LHS of all elements of $\boldsymbol{U}_i$ are the same
  - $\boldsymbol{U}_1 = \{BH \rightarrow C, BH \rightarrow K\}$, $U_2 = \{A \rightarrow D\}$,

    $\boldsymbol{U}_3 = \{C \rightarrow E\}$, $U_4 = \{L \rightarrow A\}$, $U_5 = \{E \rightarrow L\}$
- **step 3**: For each $\boldsymbol{U}_i$ form schema $\mathbf{R_i} = (R_i, \boldsymbol{U}_i)$, where R_i is the set of all attributes mentioned in $\boldsymbol{U}_i$
  - Each FD of $\boldsymbol{U}$ will be in some $\mathbf{R_i}$. Hence the decomposition is *dependency preserving*
  - $\mathbf{R_1} = (BHCK;\ BH \rightarrow C, BH \rightarrow K)$, $\mathbf{R_2} = (AD;\ A \rightarrow D)$, $\mathbf{R_3} = (CE;\ C \rightarrow E)$, $\mathbf{R_4} = (AL;\ L \rightarrow A)$, $\mathbf{R_5} = (EL;\ E \rightarrow L)$

Synthesizing a 3NF schema cont'd

- **step 4**: If no R_i is a superkey of **R,** add schema $\mathbf{R_0} = (R_0, \{\})$ where R_0 is a key of **R**.
  - $\mathbf{R_0} = (BGH, \{\})$
    - $\mathbf{R_0}$ might be needed when not all attributes are necessarily contained in $R_1 \cup R_2 \ldots \cup R_n$
      - 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)
    - $\mathbf{R_0}$ might be needed even if all attributes are accounted for in $R_1 \cup R_2 \ldots \cup R_n$
      - Example: $(ABCD; \{A \rightarrow B, C \rightarrow D\})$.

        Step 3 decomposition:

        $R_1 = (AB; \{A \rightarrow B\})$, $R_2 = (CD; \{C \rightarrow D\})$.

        Lossy! Need to add (AC; { }), for losslessness
  - Step 4 guarantees lossless decomposition.