YORK
UNIVERSITÉ
UNIVERSITY

Design Theory for Relational DBs: Functional Dependencies, Schema Decomposition, Normal Forms

EECS3421 - Introduction to Database Management Systems

Database Design Theory

- Guides systematic improvements to database schemas
- General idea:
  - Express constraints on the data
  - Use these to decompose the relations
- Ultimately, get a schema that is in a "*normal form*"
  - guarantees certain desirable properties
  - "normal" in the sense of conforming to a standard
- The process of converting a schema to a normal form is called *normalization*

Goal #1: remove redundancy

Consider this schema

Student Name	Student Email	Course	Instructor
Xiao	xiao@gmail	EECS3421	Smith
Xiao	xiao@gmail	EECS2031	Brown
Jaspreet	jaspreet@gmail	EECS3421	Smith

- What if… Xiao changes email addresses?
  - *update anomaly*: need to update more than one tuples
- What if… Xiao drops EECS2031?
  - *deletion anomaly*: loss of information that course is taught by Brown
- What if… We need to create a new course, EECS4411
  - *insertion anomaly*: how to fill rest of information (name, email, …)

Multiple relations => exponentially worse

Goal #2: expressing constraints

- Consider the following schemata:

Students(yorkid, name, email)

vs.

Students(yorkid, name)
Emails(yorkid, address)

Maybe a student has more than one emails that we would like to register (in the first schema there will be a ***redundancy****)*

- Consider also:

House(street, city, value, owner, propertyTax)

vs.

House(street, city, value, owner)
TaxRates(city, value, propertyTax)

TaxRates are defined by city, so there is no need to repeat for each single House (in the first schema there will be a ***redundancy****)*

Dependencies, constraints are domain-dependent

Overview

- Part I: Functional Dependencies
- Part II: Schema Decomposition
- Part III: Normal Forms

PART 1:
FUNCTIONAL DEPENDENCIES

Functional dependencies

- Let **X**, **Y** be sets of attributes from relation **R**
- **X** -> **Y** (we say: "**X** *functionally determines* **Y**")
  - Any tuples in **R** which agree in all attributes of **X** must also agree in all attributes of **Y**
  - Or, "The values of attributes **Y** are a function of those in **X**"
  - Not necessarily an easy function to compute, mind you

  => Consider X -> h, where h is the hash of attributes in X
- Notational conventions
  - "a", "b", "c" – specific attributes
  - "A", "B", "C" – sets of (unnamed) attributes
  - abc -> def – same as {a,b,c} -> {d,e,f}

Most common to see <u>singletons</u> (X -> y or abc -> d)

Rules and principles about FDs

- Rules
  - The splitting/combining rule
  - Trivial FDs
  - The transitive rule
- Algorithms related to FDs
  - the closure of a set of attributes of a relation
  - a minimal basis of a relation

The Splitting/Combining rule of FDs

- Attributes on right independent of each other
  - Consider *a,b,c -> d,e,f*
  - “Attributes a, b, and c functionally determine d, e, and f”

  => No mention of d relating to e or f directly
- Splitting rule (useful to split up right side of FD)
  - *abc -> def* becomes *abc -> d*, *abc -> e* and *abc -> f*
- No safe way to split left side
  - *abc -> def* is NOT the same as *ab -> def* and *c -> def*!
- Combining rule (useful to combine right sides):
  - *if abc -> d*, *abc -> e*, *abc -> f* holds, then *abc -> def* holds

Splitting FDs – example

- Consider the relation and FD
  - EmailAddress(user, domain, firstName, lastName)
  - user, domain -> firstName, lastName
- The following hold
  - user, domain -> firstName
  - user, domain -> lastName
- The following do NOT hold!
  - user -> firstName, lastName
  - domain -> firstName, lastName

Gotcha: “doesn’t hold” = “not all tuples” != “all tuples not”

Trivial FDs

- Not all functional dependencies are useful
  - *A -> A* always holds
  - *abc -> a* also always holds (right side is subset of left side)
- FD with an attribute on both sides is "trivial"
  - Simplify by removing **L** ∩ **R** from **R**

    abc -> ad becomes *abc -> d*
  - Or, in singleton form, delete trivial FDs

    abc -> a and *abc -> d* becomes just *abc -> d*

Transitive rule

- The transitive rule holds for FDs
  - Consider the FDs: *a -> b* and *b -> c*; then *a->c* holds
  - Consider the FDs: *ad -> b* and *b -> cd*; then *ad->cd* holds or just *ad->c* (because of the trivial dependency rule)

Identifying functional dependencies

- FDs are domain knowledge
  - Intrinsic features of the data you're dealing with
  - Something you know (or assume) about the data
- Database engine cannot identify FDs for you
  - Designer must specify them as part of schema
  - DBMS can only enforce FDs when told to
- DBMS cannot safely "optimize" FDs
  - It has only a finite sample of the data
  - An FD constrains the entire domain

Coincidence or FD?

ID	Email	City	Country	Surname
1983	tom@gmail.com	Toronto	Canada	Fairgrieve
8624	mar@bell.com	London	Canada	Samways
9141	scotty@gmail.com	Winnipeg	Canada	Samways
1204	birds@gmail.com	Aachen	Germany	Lakemeyer

- What if we try to infer FDs from the data?
  - ID -> email, city, country, surname
  - email -> city, country, surname
  - city -> country
  - surname -> country

Domain knowledge required to validate FDs

Keys and FDs

- Consider relation **R** with attributes **A**
- Superkey
  - Any **S** $\subseteq$ **A** s.t. *S -> A*

  => Any subset of **A** which determines all remaining attributes in **A**
- Candidate key (or key)
  - **C** $\subseteq$ **A** s.t. *C -> A* and *X -> A* does not hold for any **X** $\subset$ **C**

  => A superkey which contains no other superkeys

  => Remove any attribute from **C** and you no longer have a key
- Primary key
  - The candidate key we use to identify the relation

  => Always exists, only one allowed, doesn't matter which **C** we use
- Prime attribute
  - $\exists$ candidate key **C** s.t. **x**$\in$**C**

  => attribute that participates in at least one key

FD: relaxes the concept of a “key”

- Superkey: X -> R
  - A superkey must include all remaining attributes of the relation on the RHS (Right-Hand-Side)
- Functional dependency: X -> Y
  - An FD can involve just a subset of them
- Example:

  Houses(street, city, value, owner, tax)
  - street,city -> value,owner,tax *(both FD and key)*
  - city,value -> tax *(FD only)*

Cyclic functional dependencies?

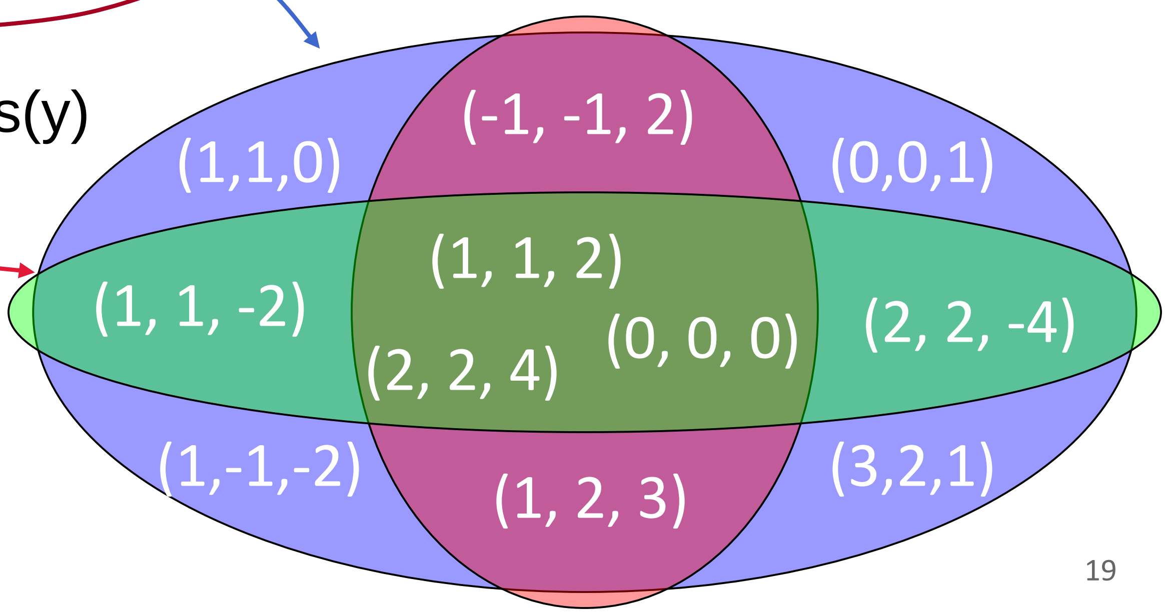
- Attributes on right side of one FD may appear on left side of another!
  - Simple example: assume relation (A, B) & FDs: A->B, B->A
  - What does this say about A and B?
- Example
  - studentID->email   email->studentID

Geometric view of FDs

- Let **D** be the domain of tuples in **R**
  - Every possible tuple is a point in **D**
- FD **X** on **R** restricts tuples in **R** to a subset of **D**
  - Points in **D** which violate **X** cannot be in **R**
- Example: D(x,y,z)
  - xy -> z

  => z = abs(x) + abs(y)
  - z -> xy

  => x=y=abs(z)/2



Inferring functional dependencies

- Problem
  - Given FDs X_1 -> a_1, X_2 -> a_2, etc.
  - Does some FD Y -> B (not given) also hold?
- Consider the dependencies

  A -> B, B -> C

  Does A -> C hold?

  Intuitively, A -> C also holds

  The given FDs entail (imply) it (transitivity rule)

How to prove it in the general case?

Closure test for FDs

- Consider relation **R**
- Given attribute set $\mathbf{A} \subseteq \mathbf{R}$ and FD set **F**
  - Denote $\mathbf{A_F}^{+}$ as the closure of **A** relative to **F**

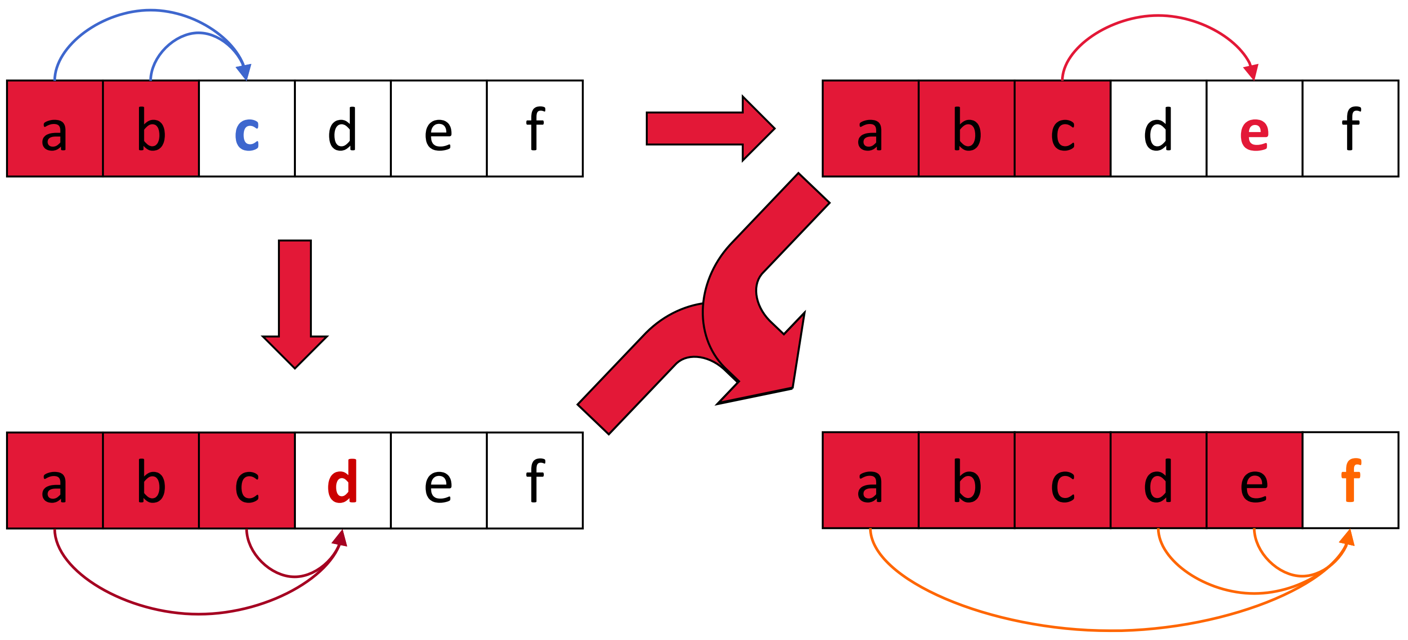
  => $\mathbf{A_F}^{+}$ = set of all FDs given or implied by **A**
- Computing the [transitive] closure of **A**
  - Start: $A_F^{+} = A$, $F' = F$
  - While $\exists X \in F'$ s.t. $LHS(X) \subseteq A_F^{+}$:

    $A_F^{+} = A_F^{+} \cup RHS(X)$

    $F' = F' - X$
  - At end: $A \rightarrow B \; \forall B \in A_F^{+}$

Closure test – example

- Consider **R**(a,b,c,d,e,f)
  with FDs set **F** = {ab -> c, ac -> d, c -> e, ade -> f }
- Find $\mathbf{A_F}^+$ if A = ab or find $\{a,b\}^+$



$\{a,b\}^+$={a,b,c,d,e,f} or *ab -> cdef -- ab is a candidate key!*

Example : Closure Test

R(A, B, C, D, E)

F: *AB -> C*
A -> D
D -> E
AC -> 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 -> E* entailed by ***F***? *Yes*

Is *D -> C* entailed by ***F***? *No*

Result: X_F^+ allows us to determine all FDs of the form *X -> Y* entailed by ***F***

Discarding redundant FDs

- Minimal basis: opposite extreme from closure
- Given a set of FDs F, want to find **minimal basis F'** s.t.
  - F' $\subseteq$ F
  - F' entails X $\forall X \in F$
- Properties of a minimal basis F'
  - RHS is always singleton
  - If any FD is removed from F', F' is no longer a minimal basis
  - If for any FD in F' we remove one or more attributes from the LHS of X $\in$ F', the result is no longer a minimal basis

Constructing a minimal basis

Straightforward but time-consuming

1. Split all RHS into singletons
2. $\forall X \in F'$, test whether $J = (F'-X)^+$ is still equivalent to F^+

=> Might make F' too small

3. $\forall i \in LHS(X)\ \forall X \in F'$, let $LHS(X') = LHS(X) - i$
   Test whether $(F'-X+X')^+$ is still equivalent to F^+

=> Might make F' too big

4. Repeat (2) and (3) until neither makes progress

Minimal Basis: Example

- Relation R: R(A, B, C, D)
- Defined FDs:
  - F = {A->AC, B->ABC, D->ABC}

Find the minimal Basis M of F

Minimal Basis: Example (cont.)

1st Step

- H = {A->A, A->C, B->A, B->B, B->C, D->A, D->B, D->C}

2nd Step

- A->A, B->B: can be removed as trivial
- A->C: can't be removed, as there is no other LHS with A
- B->A: can't be removed, because for J=H-{B->A} is B^+=BC
- B->C: can be removed, because for J=H-{B->C} is B^+=ABC
- D->A: can be removed, because for J=H-{D->A} is D^+=DBA
- D->B: can't be removed, because for J=H-{D->B} is D^+=DC
- D->C: can be removed, because for J=H-{D->C} is D^+=DBAC

Step outcome => H = {A->C, B->A, D->B}

Minimal Basis: Example (cont.)

3rd Step

- H doesn't change as all LHS in H are single attributes

4th Step

- H doesn't change

Minimal Basis: M = H = {A->C, B->A, D->B}

Caveat: Different minimal bases are possible

PART II:
SCHEMA DECOMPOSITION

FDs and redundancy

- Given relation **R** and FDs **F**
  - **R** often exhibits anomalies due to redundancy
  - **F** identifies many (not all) of the underlying problems
- Idea
  - Use **F** to identify “good” ways to split relations
  - Split **R** into 2+ smaller relations having less redundancy
  - Split up **F** into subsets which apply to the new relations (compute the projection of functional dependencies)

Schema decomposition

- Given relation R and FDs F
  - Split **R** into $\mathbf{R_i}$ s.t. $\forall$i $\mathbf{R_i} \subset \mathbf{R}$ (no new attributes)
  - Split **F** into $\mathbf{F_i}$ s.t. $\forall$i **F** entails $\mathbf{F_i}$ (no new FDs)
  - **Note:** $\mathbf{F_i}$ involves only attributes in $\mathbf{R_i}$
- Caveat: entirely possible to lose information
  - $\mathbf{F^+}$ may entail FD **X** which is not in $(\mathrm{U}_i\ \mathbf{F_i})^+$

  => Decomposition lost some FDs (*dependency not preserved*)
  - Possible to have $\mathbf{R} \subset \Join_i \mathbf{R_i}$

  => Decomposition lost some relationship (*lossy decomposition*)
- Goal: minimize anomalies without losing info

We'll revisit information loss later

Desired Properties of Decomposition

- Lossless-join
- Dependency-preserving
- Anomaly-free (no redundancies)

This may be achieved through the use of Normal Forms

Splitting relations – example

- Consider the following relation **R**:

Student Name	Student Email	Course	Instructor
Xiao	xiao@gmail	eecs3421	Smith
Xiao	xiao@gmail	eecs4411	Brown
Jaspreet	jaspreet@gmail	eecs3421	Smith

- One possible decomposition of **R**

    Students(name, email)
    Taking(email, course)
    Courses(course, instructor)

- Students $\bowtie$ Taking $\bowtie$ Courses reconstructs the right tuples!

Gotcha: lossy join decomposition

- Consider a relation **R** with one more tuple

Student Name	Student Email	Course	Instructor
Xiao	xiao@gmail	eecs3421	Smith
Xiao	xiao@gmail	eecs4411	Brown
Jaspreet	jaspreet@gmail	eecs3421	Smith
Mary	mary@gmail	eecs4411	Rosenburg

- Students $\bowtie$ Taking $\bowtie$ Courses has bogus tuples!
  - Mary is not taking Brown's section of eecs4411
  - Xiao is not taking Rosenburg's section of eecs4411

Why did this happen? How to prevent it?

Information loss with decomposition

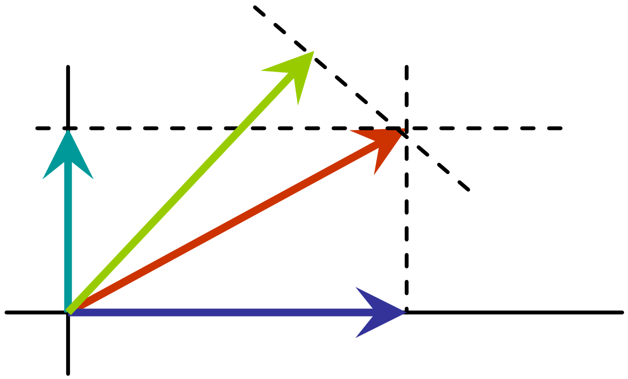
- Decompose **R** into **S** and **T**
  - Consider FD a→b, with a only in **S** and b only in **T**
- FD loss
  - Attributes a and b no longer in same relation

  => Must join **T** and **S** to enforce a→b (expensive)
- Join loss
  - LHS and RHS no longer in same relation, no other connection
  - Neither (**S** ∩ **T**) → **S** nor (**S** ∩ **T**) → **T** in F^+

  => Joining **T** and **S** produces bogus tuples (irreparable)
- In our example:
  - ({email,course} ∩ {course,instructor}) = {course}
  - course -/-> course, instructor and course -/-> email, course

Projecting FDs

- Once we've split a relation we have to refactor our FDs to match
  - Each FDs must only mention attributes from one relation
- Similar to geometric projection
  - Many possible projections (depends on how we slice it)
  - Keep only the ones we need (minimal basis)



FD projection algorithm

- Start with $\mathbf{F_i} = \varnothing$
- For each subset $\mathbf{X}$ of $\mathbf{R_i}$
  - Compute $\mathbf{X^+}$
  - For each attribute **a** in $\mathbf{X^+}$
    - If **a** is in $\mathbf{R_i}$
      - add $\mathbf{X} \rightarrow \mathbf{a}$ to $\mathbf{F_i}$
- Compute the minimal basis of $\mathbf{F_i}$
- Projection is expensive
  - Suppose $\boldsymbol{R_1}$ has *n* attributes
  - How many subsets of $\boldsymbol{R_1}$ are there?

Making projection more efficient

- Ignore trivial dependencies
  - No need to add $X \rightarrow A$ if A is in X itself
- Ignore trivial subsets
  - The empty set or the set of all attributes (both subsets of **X**)
- Ignore supersets of **X** if $X^+ = \mathbf{R}$
  - They can only give us "weaker" FDs (with more on the LHS)

45

Example: Projecting FD's

- ABC with FD's $A\text{->}B$ and $B\text{->}C$
  - $A^+=ABC$; yields $A\text{->}B$, $A\text{->}C$
    - We ignore A->A as trivial
    - We ignore the supersets of **A**, $\boldsymbol{AB}^+$ and $\boldsymbol{AC}^+$, because they can only give us "weaker" FDs (with more on the LHS)
  - $B^+=BC$; yields $B\text{->}C$
  - $C^+=C$; yields nothing.
  - $BC^+=BC$; yields nothing.

46

Example -- Continued

- Resulting FD's: *A->B*, *A->C*, and *B->C*
- Projection onto *AC* : *A->C*
  - Only FD that involves a subset of {*A*,*C*}
- Projection on BC: *B->C*
  - Only FD that involves subset of {B, C}

47

PART III:
NORMAL FORMS

Motivation for normal forms

- Identify a “good” schema
  - For some definition of “good”
  - Avoid anomalies, redundancy, etc.
- Many normal forms
  - 1st
  - 2nd
  - 3rd
  - Boyce-Codd
  - ... and several more we won’t discuss…

BCNF $\subseteq$ 3NF $\subseteq$ 2NF $\subseteq$ 1NF (focus on 3NF/BCNF)

1^{st} normal form (1NF)

- No multi-valued attributes allowed
  - Imagine storing a list/set of things in an attribute

  => Not really even expressible in RA
- Counterexample
  - Course(name, instructor, [student,email]*)
  - Redundancy in non-list attributes

Name	Instructor	Student Name	Student Email
eecs3421	Johnson	Xiao	xiao@gmail
		Jaspreet	jaspreet@utsc
		Mary	mary@utsc
eecs4411	Rosenburg	Jaspreet	jaspreet@utsc

2nd normal form (2NF)

- Non-prime attributes depend on candidate keys
  - Consider non-prime (ie. not part of a key) attribute ‘a’
  - Then ∃FD X s.t. X -> a and X is a candidate key
- Counterexample
  - Movies(<u>title</u>, <u>year</u>, <u>star</u>, studio, studioAddress, salary)
  - FD: title, year -> studio; studio -> studioAddress; star->salary

Title	Year	Star	Studio	StudioAddr	Salary
Star Wars	1977	Hamill	Lucasfilm	1 Lucas Way	$100,000
Star Wars	1977	Ford	Lucasfilm	1 Lucas Way	$100,000
Star Wars	1977	Fisher	Lucasfilm	1 Lucas Way	$100,000
Patriot Games	1992	Ford	Paramount	Cloud 9	$2,000,000
Last Crusade	1989	Ford	Lucasfilm	1 Lucas Way	$1,000,000

3rd normal form (3NF)

- Non-prime attr. depend *only* on candidate keys
  - Consider FD X -> a
  - Either a $\in$ X OR X is a superkey OR a is prime (part of a key)

  => No transitive dependencies allowed
- Counterexample:
  - studio -> studioAddr

    (*studioAddr* depends on *studio* which is not a candidate key)

Title	Year	Studio	StudioAddr
Star Wars	1977	Lucasfilm	1 Lucas Way
Patriot Games	1992	Paramount	Cloud 9
Last Crusade	1989	Lucasfilm	1 Lucas Way

3NF, dependencies, and join loss

- **Theorem**: always possible to convert a schema to lossless-join, dependency-preserving 3NF
- **Caveat**: always *possible* to create schemas in 3NF for which these properties do not hold
- FD loss example:
  - MovieInfo(title, year, studioName)
  - StudioAddress(title, year, studioAddress)

  => Cannot enforce studioName -> studioAddress
- Join loss example:
  - Movies(title, year, star)
  - StarSalary(star, salary)

  => Cannot enforce Movies $\bowtie$ StarSalary yields bogus tuples (irreparable)

3NF Synthesis Algorithm

Objective: Obtain a lossless and dependency-preserving decomposition of **R**

1. Find a minimal cover $\mathbf{F}_{\mathbf{min}}$ of **F**
2. For each LHS(X): $\forall X \in \mathbf{F}_{\mathbf{min}}$, do:
   - Create a *relation schema* using $\boldsymbol{X_{F_{min}}}^{+}$
3. Place any remaining attributes that have not been placed in any relations in step 2 in a single relation schema
4. If a key of **R** is not found in any relation , then add a trivial relation that consists of the key of **R** (if this trivial relation is useless, omit it)

3NF Synthesis Algorithm: Example

Question

Given:

- A Relation: **R**=(A, B, C, D, E, F, G, H)
- A Set of FDs **F** in **R**: **F**={A→CD, ACF→G, AD→BEF, BCG→D, CF→AH, CH→G, D→B, H→DEG}

Decompose **R** into a collection of relations $\mathbf{R_i}$ using the 3NF synthesis algorithm (which obtains a lossless and dependency-preserving decomposition of **R**)

3NF Synthesis Algorithm: Example

Answer

1. Find minimal basis of **F**:

   $\mathbf{F}_{\mathbf{min}}$ = {A→C, A→F, BCG→D, CF→A, CF→H, D→B, H→D, H→E, H→G}

2. Create relation schemas based on $\boldsymbol{X_{F_{min}}^{+}}$:

   Closures: {$A_{F_{min}}^{+}$ →ACFH, $BCG_{F_{min}}^{+}$→ BCGD, $H_{F_{min}}^{+}$→ HDEG}

   Relations: **R1**(A, C, F, H), **R2**(B, C, G, D), **R3**(H, D, E, G)

3. No remaining attributes (of **R**), thus no need to place attributes in any of the available relations

4. A key of **R** was **A** which is already in relation **R1**, so no need to add a trivial relation that consists of the key of **R**

Follow the detailed example online

Boyce-Codd normal form (BCNF)

- One additional restriction over 3NF
  - All non-trivial FD have superkey LHS
- Counterexample
  - CanadianAddress(<u>street</u>, <u>city</u>, <u>province</u>, postalCode)
  - Candidate keys: {street, postalCode}, {street, city, province}
  - FD: postalCode -> city, province

  - Satisfies 3NF: city, province both non-prime
  - Violates BCNF: postalCode is not a superkey

  => Possible anomalies involving postalCode

Do we care? How often do postal codes change?

Limits of decomposition

- Pick two…
  - Lossless-join
  - Dependency-preservation
  - Anomaly-free
- 3NF
  - Always allows join lossless and dependency preserving
  - May allow some anomalies
- BCNF
  - Always excludes anomalies
  - May give up one of lossless-join or dependency-preserving

Use domain knowledge to choose 3NF vs. BCNF

What is Next?

- Read Ullman & Widom’s textbook (Chapter 3)
- Check detailed examples on Course’s website
  - Sample 3NF Problem
  - Sample BCNF Problem
- Practice using online resources and examples