Databases and Information Systems CS303

Normalization

07-09-2023

Recap

- First Normal Form
- Functional Dependency
- Boyce Codd Normal Form
- Third Normal Form

Functional Dependency

- Let R be the set of all attributes of the relational schema r(R) and let $\alpha \subseteq$ R and $\beta \subseteq$ R. Given an instance of r,
  we say $\alpha \rightarrow \beta$ if for any two tuples t_1 and t_2 if $t_1[\alpha] = t_2[\alpha]$ then $t_1[\beta] = t_2[\beta]$

- Example: inst_dept (ID, name, salary, dept name, building, budget)
  - dept_name → building, budget
  - ID, dept_name → name, salary, building, budget

Boyce Codd Normal Form

- A relation schema R is in BCNF with respect to a set F of functional dependencies if, for all functional dependencies in F^+ of the form $\alpha \rightarrow \beta$, where $\alpha \subseteq R$ and $\beta \subseteq R$: at least one of the following holds:
  - $\alpha \rightarrow \beta$ is a trivial functional dependency (that is, $\beta \subseteq \alpha$)
  - α is a superkey for schema R

- A database design is in BCNF if every relational schema of the design is in BCNF

Boyce Codd Normal Form : General decomposition Rule

- Let r(R) be a schema that is not in BCNF
  (There is a non-trivial functional dependency $\alpha \rightarrow \beta$ where α is not a superkey)

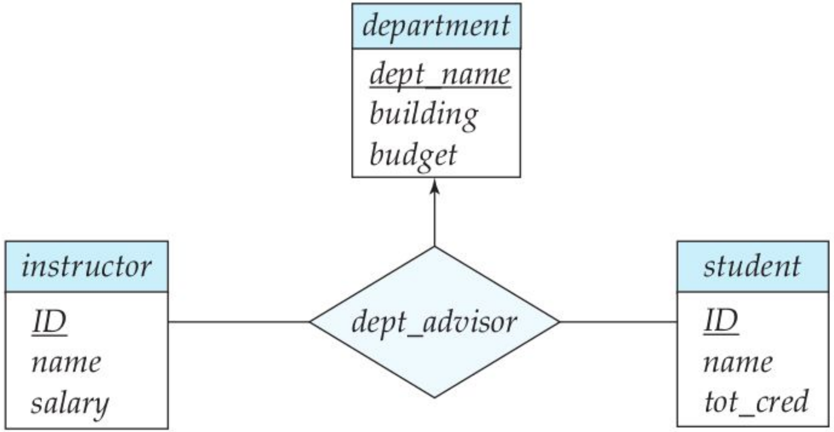
- Replace r(R) with two new schemas
  - $\alpha \cup \beta$
  - $(R - (\beta - \alpha))$

- Example: inst_dept (ID, name, salary, dept_name, building, budget)
  - dept_name → building, budget

- Application of the rule might result in smaller relations that are not in BCNF
  - Repeat for procedure for smaller relations till the resulting schema is in BCNF

Boyce Codd Normal Form : Dependency Preservation

- dept_advisor (s_ID, i_ID, dept_name)
  - s_ID, dept_name → i_ID
  - i_ID → dept_name
    (every instructor can be advisor
    for a single department)
  - Not in BCNF because i_ID is not superkey
  - (s_ID, i_ID)
  - (i_ID, dept_name)
  - s_ID, dept_name → i_ID
    decomposition makes it computationally hard to enforce this functional dependency



- BCNF is not dependency preserving
  - 3NF (weaker than BCNF) is always dependency preserving

Third Normal Form

- A relation schema r(R) is in Third Normal Form (3NF) with respect to a set F of functional dependencies if, for all functional dependencies in F^+ of the form $\alpha \rightarrow \beta$, where $\alpha \subseteq R$ and $\beta \subseteq R$: at least one of the following holds:
    - $\alpha \rightarrow \beta$ is a trivial functional dependency (that is, $\beta \subseteq \alpha$).
    - α is a superkey for schema R.
    - Each attribute A in $\beta - \alpha$ is contained in some candidate key of R
- Each such A can be part of different candidate keys
- Any schema that is in BCNF is also in 3NF

Third Normal Form

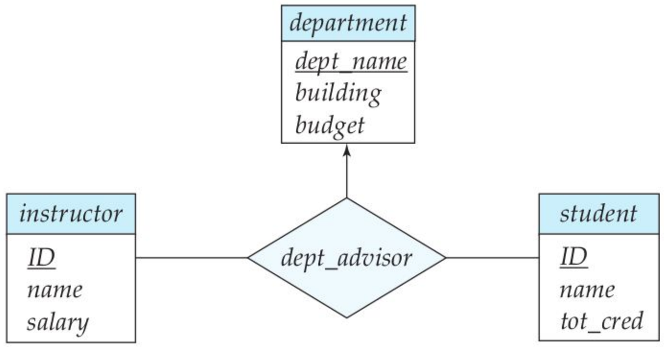
- A relation schema r(R) is in Third Normal Form (3NF) with respect to a set F of functional dependencies if, for all functional dependencies in F^+ of the form $\alpha \rightarrow \beta$, where $\alpha \subseteq$ R and $\beta \subseteq$ R : at least one of the following holds:
  - $\alpha \rightarrow \beta$ is a trivial functional dependency (that is, $\beta \subseteq \alpha$).
  - α is a superkey for schema R.
  - Each attribute A in β - α is contained in some candidate key of R

- dept_advisor (s_ID, i_ID, dept_name) is in 3NF

  - s_ID, dept_name $\rightarrow$ i_ID
  - i_ID $\rightarrow$ dept_name

    (every instructor can be advisor

    for a single department)



Decomposition Algorithms

- Real life schema diagrams are typically Huge
- Cannot be done manually
- We need Algorithms that take a schema as an input and decompose them into BCNF or 3NF
- For that we need subroutines to compute F^+, Check if a set of attributes is a superkey

Computing F^+

- Testing BCNF and 3NF needs F^+ obtained from the initial set of functional dependencies F

- Suppose r (A, B, C, G, H, I) is a relational schema

$$A \rightarrow B$$
$$A \rightarrow C$$
$$CG \rightarrow H$$
$$CG \rightarrow I$$
$$B \rightarrow H$$

$$A \rightarrow H$$

Computing F^+

- Use the following Axioms to compute F^+ (Armstrong's axioms)
  - Reflexivity Rule : If $\beta \subseteq \alpha$ then $\alpha \rightarrow \beta$
  - Augmentation Rule : If $\alpha \rightarrow \beta$ and γ is a set of attributes then $\gamma\alpha \rightarrow \gamma\beta$
  - Transitive Rule : If $\alpha \rightarrow \beta$ and $\beta \rightarrow \gamma$ then $\alpha \rightarrow \gamma$

- Armstrong's axioms are sound

  (They do not generate any incorrect functional dependencies)

- Armstrong's axioms are complete

  (All functional dependencies can be deduced using these rules)

Computing F^+

- Use the following Axioms to compute F^+ (Armstrong's axioms)
  - Reflexivity Rule : If $\beta \subseteq \alpha$ then $\alpha \rightarrow \beta$
  - Augmentation Rule : If $\alpha \rightarrow \beta$ and γ is a set of attributes then $\gamma\alpha \rightarrow \gamma\beta$
  - Transitive Rule : If $\alpha \rightarrow \beta$ and $\beta \rightarrow \gamma$ then $\alpha \rightarrow \gamma$

$F^+ = F$
repeat
    for each functional dependency f in F^+
        apply reflexivity and augmentation rules on f
        add the resulting functional dependencies to F^+
    for each pair of functional dependencies f_1 and f_2 in F^+
        if f_1 and f_2 can be combined using transitivity
            add the resulting functional dependency to F^+
until F^+ does not change any further

- Correctness of the algorithm ?
- Running time of the algorithm ?

Computing F^+

- Use the following Axioms to compute F^+ (Armstrong's axioms)
  - Reflexivity Rule : If $\beta \subseteq \alpha$ then $\alpha \rightarrow \beta$
  - Augmentation Rule : If $\alpha \rightarrow \beta$ and γ is a set of attributes then $\gamma\alpha \rightarrow \gamma\beta$
  - Transitive Rule : If $\alpha \rightarrow \beta$ and $\beta \rightarrow \gamma$ then $\alpha \rightarrow \gamma$

- Other userful rules (Can be derived using the above axioms)
  - Union Rule : If $\alpha \rightarrow \beta$ and $\alpha \rightarrow \gamma$ then $\alpha \rightarrow \beta\gamma$
  - Decomposition Rule : If $\alpha \rightarrow \beta\gamma$ then $\alpha \rightarrow \beta$ and $\alpha \rightarrow \gamma$ holds
  - Pseudotransitivity Rule: If $\alpha \rightarrow \beta$ and $\beta\gamma \rightarrow \delta$ then $\alpha\gamma \rightarrow \delta$

Closure of Attribute set

- An Attribute B is functionally dependent on α if $\alpha \rightarrow B$
- To check if α is a superkey, we should compute the set of all attributes functionally dependant on α
- One way to do this is to compute F^+ , take all FDs with α as left-hand side and right-hand side will be the union of the right-hand side of all such FDs
    - This is expensive

Closure of Attribute set

- There is an efficient algorithm

$$
\begin{array}{l}
\mathit{result} := \alpha; \\
\textbf{repeat} \\
\quad \textbf{for each } \text{functional dependency } \beta \rightarrow \gamma \textbf{ in } F \textbf{ do} \\
\quad\quad \textbf{begin} \\
\quad\quad\quad \textbf{if } \beta \subseteq \mathit{result} \textbf{ then } \mathit{result} := \mathit{result} \cup \gamma; \\
\quad\quad \textbf{end} \\
\textbf{until } (\mathit{result} \text{ does not change})
\end{array}
$$

$$
\begin{array}{l}
A \rightarrow B \\
A \rightarrow C \\
CG \rightarrow H \\
CG \rightarrow I \\
B \rightarrow H
\end{array}
$$

- Over schema r(A,B,C,G,H,I) Compute $(AG)^+$
  - result = {A,G}
  - result = {A,G,B}
  - result = {A,G,B,C}
  - result = {A,G,B,C,H}
  - result = {A,G,B,C,H,I}

Closure of Attribute set

- There is an efficient algorithm

$$
\begin{array}{l}
\mathit{result} := \alpha; \\
\textbf{repeat} \\
\quad\quad \textbf{for each} \text{ functional dependency } \beta \rightarrow \gamma \textbf{ in } F \textbf{ do} \\
\quad\quad\quad \textbf{begin} \\
\quad\quad\quad\quad \textbf{if } \beta \subseteq \mathit{result} \textbf{ then } \mathit{result} := \mathit{result} \cup \gamma; \\
\quad\quad\quad \textbf{end} \\
\textbf{until } (\mathit{result} \text{ does not change})
\end{array}
$$

- Correctness of the Algorithm
    - Soundness and Completeness

- Running time of the Algorithm

Closure of Attribute set : Uses

- To test if α is a superkey
- Check if a functional dependency $\alpha \rightarrow \beta$ holds
- Alternative way to compute F^+

Canonical Cover

- Suppose database has functional dependency set F:

- Whenever a database is updated, the system should ensure all functional dependencies are satisfied

- If we have a minimal set of functional dependencies F^* that imply all the functional dependencies of F then the check can be made faster

Canonical Cover

- An attribute of a functional dependency is extraneous if we can remove it without changing the closure set of the functional dependencies.

- Consider a set of functional dependencies F and let $\alpha \rightarrow \beta$ be in F
  - Attribute A in α is extraneous if F logically implies (F - $\{\alpha \rightarrow \beta\}$) U { ($\alpha$ - A) $\rightarrow \beta$ }
  - Attribute A in β is extraneous if the set (F - $\{\alpha \rightarrow \beta\}$) U { $\alpha \rightarrow$ (β - A) } logically implies F

Canonical Cover

- An attribute of a functional dependency is extraneous if we can remove it without changing the closure set of the functional dependencies.

- Consider a set of functional dependencies F and let $\alpha \rightarrow \beta$ be in F
    - Attribute A in α is extraneous if F logically implies (F - $\{\alpha \rightarrow \beta\}$) U { ($\alpha$ - A) $\rightarrow \beta$ }
    - Attribute A in β is extraneous if the set (F - $\{\alpha \rightarrow \beta\}$) U { $\alpha \rightarrow$ (β - A) } logically implies F

- Example:
    - In F = {AB → C , A → C}, B is extraneous in AB → C
    - In F = {AB → CD, AB → C}   C is extraneous in AB → CD

- Note: In both cases the other direction of the implication trivially holds always

Canonical Cover : Checking for extraneous attributes

- Consider a set of functional dependencies F and let $\alpha \rightarrow \beta$ be in F
  - Attribute A in α is extraneous if F logically implies (F - $\{\alpha \rightarrow \beta\}$) U { ($\alpha$ - A) $\rightarrow \beta$ }
  - Attribute A in β is extraneous if the set (F - $\{\alpha \rightarrow \beta\}$) U { $\alpha \rightarrow$ (β - A) } logically implies F

- For a relational schema R and functional dependencies F and $\alpha \rightarrow \beta$ in F,
  Let A be an attribute in $\alpha \rightarrow \beta$

  - If A is in β : To check if A is extraneous, consider the set of functional dependencies: F* = (F - $\{\alpha \rightarrow \beta\}$) U { $\alpha \rightarrow$ (β - A) } and check if $\alpha \rightarrow \beta$ can be inferred from F* (By computing α^+ under F*)

  - If A is in α : To check if A is extraneous, consider $\gamma = \alpha$ - {A} and check if $\gamma \rightarrow \beta$ can be inferred under F (By computing γ^+)

Canonical Cover

- Canonical Cover of a set of functional dependencies F is given by F_c such that F logically implies F_c and vice-versa, such that the following conditions hold:
  - No functional dependencies in F_c contains extraneous attributes
  - Each left side of a functional dependency in F_c is unique
    (There are no two dependencies $\alpha_1 \rightarrow \beta_1$ and $\alpha_2 \rightarrow \beta_2$ in F_c such that $\alpha_1 = \alpha_2$)

$F_c = F$
repeat
    Use the union rule to replace any dependencies in F_c of the form
        $\alpha_1 \rightarrow \beta_1$ and $\alpha_1 \rightarrow \beta_2$ with $\alpha_1 \rightarrow \beta_1 \beta_2$.
    Find a functional dependency $\alpha \rightarrow \beta$ in F_c with an extraneous
        attribute either in α or in β.
        /* Note: the test for extraneous attributes is done using F_c, not F */
    If an extraneous attribute is found, delete it from $\alpha \rightarrow \beta$ in F_c.
until (F_c does not change)

Canonical Cover

- Canonical Cover of a set of functional dependencies F is given by F_c such that F logically implies F_c and vice-versa, such that the following conditions hold:
    - No functional dependencies in F_c contains extraneous attributes
    - Each left side of a functional dependency in F_c is unique
      (There are no two dependencies $\alpha_1 \rightarrow \beta_1$ and $\alpha_2 \rightarrow \beta_2$ in F_c such that $\alpha_1 = \alpha_2$)

- F and F_c have same closures.

- Testing if F is satisfied is equivalent to testing if F_c is satisfied

- It is cheaper to test F_c than F itself

Canonical Cover : Example

- Let r(A, B, C) be a relational schema with the following functional dependencies:

$$\begin{aligned} A &\rightarrow BC \\ B &\rightarrow C \\ A &\rightarrow B \\ AB &\rightarrow C \end{aligned}$$

- Step 0: $F_c = \{ A \rightarrow BC, B \rightarrow C, A \rightarrow B, AB \rightarrow C \}$
- Step 1 : Combine 1st and 3rd FD and obtain $F_c = \{ A \rightarrow BC, B \rightarrow C, AB \rightarrow C \}$
- Step 2: A is extraneous in $AB \rightarrow C$
  - because F_c implies $\{ B \rightarrow C, A \rightarrow BC \} \cup \{ B \rightarrow C \}$
    - $F_c = \{ A \rightarrow BC, B \rightarrow C \}$
- Step 3 : C is extraneous in $A \rightarrow BC$
  - Because $\{ A \rightarrow B, B \rightarrow C \}$ logically implies $A \rightarrow BC$
    - $F_c = \{ A \rightarrow B , B \rightarrow C \}$

Dependency Preservation

- Let F be a set of functional dependencies on R and let R_1 R_2..... R_n be a decomposition of R
- Restriction of F to R_i is the set F_i is the set of all functional dependencies that includes only the attributes of R_i
- Example: If r(A,B,C) is a schema with $F = \{ A \rightarrow B, B \rightarrow C, A \rightarrow C \}$ and the decomposition is
  $R_1 = (A,B)$ and $R_2 = (A,C)$
  Then $F_1 = \{ A \rightarrow B \}$ and $F_2 = \{ A \rightarrow C \}$
- Decomposition of R to R_1 R_2..... R_n is said to be dependency preserving if:

$$F^+ = (F_1 \cup F_2 \cup \ldots \cup F_m)^+$$

compute F^+;
for each schema R_i in D **do**
  begin
    $F_i :=$ the restriction of F^+ to R_i;
  end
$F' := \emptyset$
for each restriction F_i **do**
  begin
    $F' = F' \cup F_i$
  end
compute F'^+;
if $(F'^+ = F^+)$ **then** return (true)
    else return (false);

Decomposition Algorithms : BCNF

- A relation schema R is in BCNF with respect to a set F of functional dependencies if, for all functional dependencies in F^+ of the form $\alpha \rightarrow \beta$, where $\alpha \subseteq R$ and $\beta \subseteq R$: at least one of the following holds:
    - $\alpha \rightarrow \beta$ is a trivial functional dependency (that is, $\beta \subseteq \alpha$)
    - α is a superkey for schema R

- Testing for BCNF
    - Goal is to see if there exists a non-trivial $\alpha \rightarrow \beta$ in F^+ where α is not a superkey
        - For every non-trivial $\alpha \rightarrow \beta$ in F^+ Compute α^+ and see if it contains all the attributes of R
    - There are Other clever ways to do it that avoids computing F^+

Decomposition Algorithms : BCNF

- A relation schema R is in BCNF with respect to a set F of functional dependencies if, for all functional dependencies in F^+ of the form $\alpha \rightarrow \beta$, where $\alpha \subseteq$ R and $\beta \subseteq$ R : at least one of the following holds:
  - $\alpha \rightarrow \beta$ is a trivial functional dependency (that is, $\beta \subseteq \alpha$)
  - α is a superkey for schema R

- BCNF Decomposition Algorithm

```
result := {R};
done := false;
compute F+;
while (not done) do
    if (there is a schema Ri in result that is not in BCNF)
      then begin
               let α → β be a nontrivial functional dependency that holds
               on Ri such that α → Ri is not in F+, and α ∩ β = Ø;
               result := (result − Ri) ∪ (Ri − β) ∪ ( α, β);
           end
      else done := true;
```

Decomposition Algorithms : BCNF

- Example:
  class (course_id, title, dept_name, credits, sec_id, semester, year, building, room_number, capacity, time_slot_id)

- course_id → title, dept_name, credits
- building, room_number → capacity
- course_id, sec_id, semester, year → building, room_number, time_slot_id

- Candidate Key = { course_id, sec_id, semester, year }

- course_id → title, dept_name, credits (is non-trivial FD and course_id is not superkey)
  - course (course_id , title, dept_name, credits)
  - class1 (course_id, sec_id, semester, year, building, room_number, capacity, time_slot_id)

- building, room_number → capacity (is non-trivial FD and course_id is not superkey)
  - course (course_id , title, dept_name, credits)
  - class11 (building, Room_number, capacity)
  - class12 (course_id, sec_id, year, time_slot_id)

- All 3 are in BCNF (This is lossless and dependency preserving decomposition)
- Algorithm ensures losslessness but dependency preservation might be violated

Decomposition Algorithms : 3NF

- A relation schema r(R) is in Third Normal Form (3NF) with respect to a set F of functional dependencies if, for all functional dependencies in F^+ of the form $\alpha \rightarrow \beta$, where $\alpha \subseteq$ R and $\beta \subseteq$ R : at least one of the following holds:
  - $\alpha \rightarrow \beta$ is a trivial functional dependency (that is, $\beta \subseteq \alpha$).
  - α is a superkey for schema R.
  - Each attribute A in β - α is contained in some candidate key of R

let F_c be a canonical cover for F;
$i := 0$;
for each functional dependency $\alpha \rightarrow \beta$ in F_c
    $i := i + 1$;
    $R_i := \alpha\,\beta$;
if none of the schemas R_j, $j = 1, 2, \ldots, i$ contains a candidate key for R
  then
    $i := i + 1$;
    $R_i :=$ any candidate key for R;
/* Optionally, remove redundant relations */
repeat
    if any schema R_j is contained in another schema R_k
      then
        /* Delete R_j */
        $R_j := R_i$;
        $i := i - 1$;
until no more R_js can be deleted
return $(R_1, R_2, \ldots, R_i)$

Decomposition Algorithms : 3NF

- Example:
  dept_advisors (s_ID, i_ID, dept_name)

- i_ID → dept_name
- s_ID, dept_name → i_ID

- $F = F_c$ (no extraneous attributes)

- R_1 (i_ID, dept_name)
- R_2 (s_ID, dept_name, i_ID)

- Delete R_1 and retain only R_2

- Algorithm results in Lossless and Dependency preserving 3NF decomposition

```
let F_c be a canonical cover for F;
i := 0;
for each functional dependency α → β in F_c
    i := i + 1;
    R_i := α β;
if none of the schemas R_j, j = 1, 2, ..., i contains a candidate key for R
  then
    i := i + 1;
    R_i := any candidate key for R;
/* Optionally, remove redundant relations */
repeat
    if any schema R_j is contained in another schema R_k
      then
        /* Delete R_j */
        R_j := R_i;
        i := i - 1;
until no more R_js can be deleted
return (R_1, R_2, ..., R_i)
```

Comparing BCNF and 3NF

- It is always possible to obtain 3NF without sacrificing dependency preservation and losslessness
  - But might lead to having many nulls and repetition of information
- Goal:
  - BCNF + Lossless decomposition + Dependency Preservation
  - All 3 are not possible simultaneously
  - Choose BCNF or Dependency Preservation + 3NF
- SQL does not allow specification of FDs (except primary key and super keys using unique)
  - Can be done using assertions

Multivalued dependencies

- In some cases BCNF does not reduce repetitions

- Suppose one instructor can be associated with multiple departments. Moreover Instructors can have several addresses (office, current address, permanent address..)
  - inst (ID, dept_name, name, street, city)
  - FD : ID → name (but ID is not the key for inst)

  - r_1 (ID, dept_name, street, city) and r_2 (ID, name)

- Decomposition is in BCNF but there still can be repetition in r_1
  - r_{11} (ID, dept_name) and r_{12} (ID, street, city)
- How to identify this in the absence of additional constraints ?

Multivalued dependencies

- A relation schema r(R) be a relational schema and let $\alpha \subseteq R$ and $\beta \subseteq R$.
  The multivalued dependency $\alpha \twoheadrightarrow \beta$ holds if for every legal instance r of r(R) and tuples t_1 and t_2 of r such that $t_1[\alpha] = t_2[\alpha]$ then there exists t_3 and t_4 such that
  - $t_1[\alpha] = t_2[\alpha] = t_3[\alpha] = t_4[\alpha]$
  - $t_3[\beta] = t_1[\beta]$
  - $t_3[R - \beta] = t_2[R - \beta]$
  - $t_4[\beta] = t_2[\beta]$
  - $t_4[R - \beta] = t_1[R - \beta]$

	α	β	$R - \alpha - \beta$
t_1	$a_1 \ldots a_i$	$a_{i+1} \ldots a_j$	$a_{j+1} \ldots a_n$
t_2	$a_1 \ldots a_i$	$b_{i+1} \ldots b_j$	$b_{j+1} \ldots b_n$
t_3	$a_1 \ldots a_i$	$a_{i+1} \ldots a_j$	$b_{j+1} \ldots b_n$
t_4	$a_1 \ldots a_i$	$b_{i+1} \ldots b_j$	$a_{j+1} \ldots a_n$

- Multivalued dependencies are part of more general tuple generating dependencies
- Functional dependencies are part of more general equality generating dependencies

Multivalued dependencies : Closure

- If D is a set of FDs + MVDs then let D^+ denote the closure of D that contains all the dependencies that are logically implied by D

- Computed using axioms. Sample axioms:
  - If $\alpha \rightarrow \beta$ then $\alpha \twoheadrightarrow \beta$
  - If $\alpha \twoheadrightarrow \beta$ then $\alpha \twoheadrightarrow R - \alpha - \beta$

Multivalued dependencies : Fourth Normal Form

- MVD $\alpha \twoheadrightarrow \beta$ over r(R) is trivial if $\beta \subseteq \alpha$ or $\alpha \cup \beta = R$

- A relational schema r(R) is in Fourth Normal Form (4NF) with respect to a set D of FD + MVD if for all MVDs $\alpha \twoheadrightarrow \beta$ in D^+ where $\alpha \subseteq R$ and $\beta \subseteq R$ one of the following holds:
  - $\alpha \twoheadrightarrow \beta$ is a trivial MVD
  - α is a superkey of R

- A database design in in 4NF if every relational schema is in 4NF
- Note that 4NF implies BCNF

Multivalued dependencies : 4NF decomposition

- Preverves Losslessness
- But does not preserve dependencies

result := $\{R\}$;
done := false;
compute D^+; Given schema R_i, let D_i denote the restriction of D^+ to R_i
while (**not** *done*) **do**
  if (there is a schema R_i in *result* that is not in 4NF w.r.t. D_i)
    then begin
      let $\alpha \twoheadrightarrow \beta$ be a nontrivial multivalued dependency that holds on R_i such that $\alpha \rightarrow R_i$ is not in D_i, and $\alpha \cap \beta = \emptyset$;
      result := $(result - R_i) \cup (R_i - \beta) \cup (\alpha, \beta)$;
    end
  else *done* := true;

Higher Normal Forms

- Join dependencies generalize multivalued dependencies
    - Leads to Project-Join Normal Form (5NF)

- Domain Key Normal Form

- Disadvantages of higher normal forms:
    - Hard to reason
    - No sound and complete axiom set

Database Design and Normalization

- Generated while converting an E-R diagram into a relational schema
  - Good E-R diagram does not need too much normalization when translated to relational schemas
  - Relationship sets involving more than two entities can result in non-normalized schema
  - Universal-relation approach : Design one big relational schema that contains everything and then normalize

- Naming Relationship and Attributes
  - Unique Role assumption : Every name of attribute has a unique meaning
  - Convention to designate primary key attributes at the beginning
  - Prefix helps in disambiguation : student_ID, instructor_ID

Database Design and Normalization

- Denormalization for Performance
  - Occasionally designers choose to retain redundancy
  - Improves performance
    (Normalized databases need join operations to evaluate queries)
  - Alternative is to use normalized database and store denormalized database as materialized views

Database Design and Normalization

- Some aspects not considered in normalization process
- Example:
  - Store number of instructors in each department in various years

    (dept_name, year, num_inst)

    with dept_name, year → num_inst as FD
  - Alternative is to have one table for each year

    inst_count_2020 (dept_name num_inst) inst_count_2021 (dept_name, num_inst)

    With dept_name → num_inst as FD in each table
  - Another Alternative

    dept_year (dept_name, num_inst_2020, num_inst_2021)

    With dept_name → num_inst_2020, num_inst_2021

    Close to spreadsheet representation

- Last 2 alternatives are bad design choices though they are in BCNF

Modelling Temporal Data

- Temporal Data is associated with a time interval where the data is valid (called snapshot)
- Functional dependency should take temporal aspect into account
- Example : (ID, street, city, from, to)
- Temporal functional dependency : $X \xrightarrow{T} Y$
  - Makes the design and functional dependence analysis complicated
  - First design without temporal data (by considering a snapshot) then decide what temporal constraints are needed
- What should be the end time if it is currently ongoing?
  - Null ? (Problem with primary key)
  - Something far in future
- Relationship between time dependent and time independent data
  - Advisor relationship and department budget (though depends on current budget)
  - Solution: Not to add time in main relation, but maintain history separately

Reference:

Database System Concepts by Silberschatz, Korth and Sudarshan

(6th edition)

Chapter 8