Normalization

Dr. Shrutilipi Bhattacharjee, Assistant Professor, Dept. of IT, NIT Karnataka, India

Normalization or Schema Refinement

- *Normalization* or *Schema Refinement* is a technique of organizing the data in the database

- It is a systematic approach of decomposing tables to eliminate data redundancy and undesirable characteristics
  - Insertion anomalies
  - Update anomalies
  - Deletion anomalies

- Most common technique for the *schema refinement* is decomposition
  - ***Goal of Normalization:*** Eliminate redundancy

- *Redundancy* refers to repetition of same data or duplicate copies of same data stored in different locations

- *Normalization* is used for mainly two purposes:
  - Eliminating redundant (useless) data
  - Ensuring data dependencies make sense, that is data is logically stored

Anomalies

- **Update anomaly**: Employee 519 is shown as having different addresses on different records

Employees' Skills

Employee ID	Employee Address	Skill
426	87 Sycamore Grove	Typing
426	87 Sycamore Grove	Shorthand
519	94 Chestnut Street	Public Speaking
519	96 Walnut Avenue	Carpentry

- **Deletion anomaly**: All information about Dr. Giddens is lost if he or she temporarily ceases to be assigned to any courses

- **Resolution: *Decompose the schema***
  - ***Update:*** (Employee ID, Employee Address), (Employee ID, Skill)
  - ***Insert/Delete:*** (Faculty ID, Faculty Name, Faculty Hire Date), and (Faculty ID, Course Code)

- **Insertion anomaly**: Until the new faculty member, Dr. Newsome, is assigned to teach at least one course, his or her details cannot be recorded

Faculty and Their Courses

Faculty ID	Faculty Name	Faculty Hire Date	Course Code
389	Dr. Giddens	10-Feb-1985	ENG-206
407	Dr. Saperstein	19-Apr-1999	CMP-101
407	Dr. Saperstein	19-Apr-1999	CMP-201

424	Dr. Newsome	29-Mar-2007	?

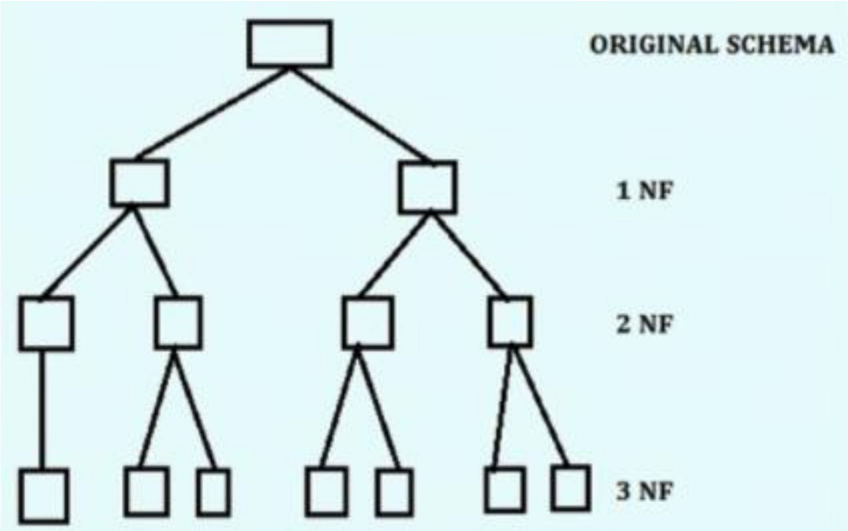
Faculty and Their Courses

Faculty ID	Faculty Name	Faculty Hire Date	Course Code
389	Dr. Giddens	10-Feb-1985	ENG-206
407	Dr. Saperstein	19-Apr-1999	CMP-101
407	Dr. Saperstein	19-Apr-1999	CMP-201

DELETE

Desirable Properties of Decomposition

- **Lossless join decomposition property**
  - It should be possible to reconstruct the original table
- **Dependency preserving property**
  - No functional dependencies (or other constraints) should get violated



Normalization and Normal Forms

- A normal form specifies a set of conditions that the relational schema must satisfy in terms of its constrains - they offer varied level of guarantee of the design

- Normalization rules are divided into various normal forms

- Most common normal forms are:
  - First normal form (1NF)
  - Second normal form (2NF)
  - Third normal form (3NF)

- Informally, a relational database relation is often described as “normalized” if it meets third normal form

- Most 3NF relations are free of insertion, deletion, and update anomalies

Normalization and Normal Forms

- Additional normal forms:
  - Elementary key normal form (EKNF)
  - Boyce codd normal form (BCNF)
  - Multivalued dependencies and Forth normal form (4NF)
  - Essential tuple normal form (ETNF)
  - Join dependencies and Fifth normal form (5NF)
  - Sixth normal form (6NF)
  - Domain/key normal form (DKNF)

1^{st} Normal Form	No repeating data groups
2^{nd} Normal Form	No partial key dependency
3^{rd} Normal Form	No transitive dependency
Boyce-Codd Normal Form	Reduce keys dependency
4^{th} Normal Form	No multi-valued dependency
5^{th} Normal Form	No join dependency

$$1NF \supset 2NF \supset 3NF \supset BCNF \supset 4NF \supset 5NF$$

First Normal Form (1NF)

- A relation is in first normal form if and only if all the underlying domains contains atomic (indivisible) values only
- In other words, a relation doesn't have multi-valued attributes (MVA)

- **Example:**
- Student(SID, Sname, Cname)

SID	Sname	Cname
S1	A	C, C++
S2	B	C++, DB
S3	A	DB

SID: Primary key
MVA exists ⇒ Not in 1NF

SID	Sname	Cname
S1	A	C
S1	A	C++
S2	B	C++
S2	B	DB
S3	A	DB

SID: Primary key
No MVA exists ⇒ In 1NF

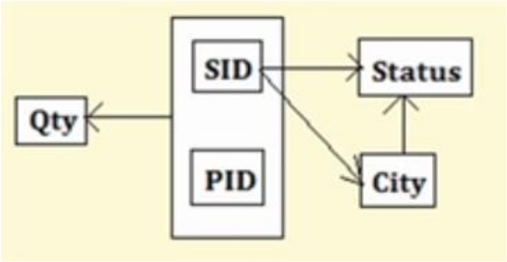
First Normal Form (1NF): Possible Redundancy

- **Example:**
- Supplier(SID, Status, City, PID, Qty)

Supplier

SID	Status	City	PID	Qty
S1	30	Delhi	P1	100
S1	30	Delhi	P2	125
S1	30	Delhi	P3	200
S1	30	Delhi	P4	130
S2	10	Karnal	P1	115
S2	10	Karnal	P2	250
S3	40	Rohtak	P1	245
S4	30	Delhi	P4	300
S4	30	Delhi	P5	315

Key: (SID, PID)



Deletion Anomaly – If we delete the tuple <S3, 40, Rohtak, P1, 245>, then we loose the information about S3 that S3 lives in Rohtak

Insertion Anomaly – We cannot insert a supplier S5 located in Karnal, until S5 supplies atleast one part

Updation Anomaly – If supplier S1 moves from Delhi to Kanpur, then it is difficult to update all the tuples containing (S1, Delhi), as SID and City respectively

Normal forms are the methods of reducing redundancy, however, sometimes 1NF increases redundancy
It doesn't make any effort in order to decrease redundancy

First Normal Form (1NF): Possible Redundancy

- **When LHS is not a Superkey:**
  - Let $X \rightarrow Y$ is a non-trivial FD over R with X is not a superkey of R, then redundancy exists between X and Y attribute set
  - Hence, in order to identify the redundancy, we need not look at the actual data, it can be identified by given functional dependency
  - Example: $X \rightarrow Y$ and X is not a *candidate key*
    ⇒ X can duplicate
    ⇒ Corresponding Y value would duplicate too

X	*Y*
1	3
1	3
2	3
2	3
4	6

- **When LHS is a Superkey:**
  - Let $X \rightarrow Y$ is a non-trivial FD over R with X is a superkey of R, then redundancy does not exist between X and Y attribute set
  - Example: $X \rightarrow Y$ and X is a *candidate key*
    ⇒ X cannot duplicate
    ⇒ Corresponding Y value may or may not duplicate

X	*Y*
1	4
2	6
3	4

Second Normal Form (2NF)

- A relation ***R*** is in the second normal form (2NF) iff:
  - ***R*** should be in first normal form
  - ***R*** should not contain any partial dependency

Partial Dependency:

Let ***R*** be a relational schema and ***X, Y, A*** be the attribute sets over ***R*** where ***X***: Any *candidate key*, ***Y***: Proper subset of *candidate key*, and ***A***: Non key attribute

If $Y \rightarrow A$ exists in ***R***, then ***R*** is not in 2NF

($Y \rightarrow A$) is a partial dependency if

- ***Y***: Proper subset of *candidate key*
- ***A***: Nonprime attribute

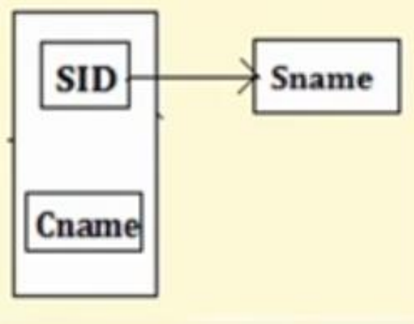
Second Normal Form (2NF)

- **Example:**
  – STUDENT(SID, Sname, Cname) (Already in 1NF)

STUDENT

SID	Sname	Cname
S1	A	C
S1	A	C++
S2	B	C++
S2	B	DB
S3	A	DB

(SID, Cname): Primary key
No MVA exists ⇒ In 1NF



Functional dependencies:
{SID, Cname} → Sname
SID → Sname

Partial dependencies:
SID → Sname (as SID is a proper subset of *candidate key* {SID,Cname})

Post normalization

R1

SID	Sname
S1	A
S2	B
S3	A

(SID): Primary key

R2

SID	Cname
S1	C
S1	C++
S2	C++
S2	DB
S3	DB

(SID, Cname): Primary key

The above two relations **R1** and **R2** are:
- Lossless join
- In 2NF
- Dependency preserving

Second Normal Form (2NF): Possible Redundancy

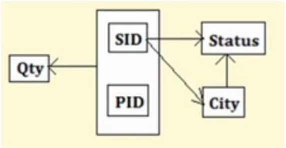
- **Example:**
- Supplier(SID, Status, City, PID, Qty)

Supplier				
SID	**Status**	**City**	**PID**	**Qty**
S1	30	Delhi	P1	100
S1	30	Delhi	P2	125
S1	30	Delhi	P3	200
S1	30	Delhi	P4	130
S2	10	Karnal	P1	115
S2	10	Karnal	P2	250
S3	40	Rohtak	P1	245
S4	30	Delhi	P4	300
S4	30	Delhi	P5	315
Key: (SID, PID)				

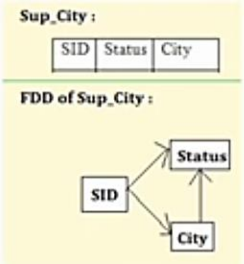
Partial dependencies:

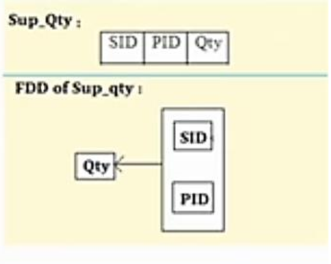
SID → Status

SID → City



Post normalization





Deletion Anomaly – If we delete a tuple in the **Sup_City** relation, then we not only loose the information about supplier, but also loose the status value of a particular city

Insertion Anomaly – We cannot insert a city and its status until a supplier supplies atleast one part

Updation Anomaly – If the status value for a city is changed, then we will face the problem of searching every tuple for that city

Third Normal Form (3NF)

- Let **R** be the relational schema
- [E. f. Codd, 1971] **R** is in 3NF only if:
  - **R** should be in 2NF
  - **R** should not contain *transitive dependencies* (OR, every non-prime attribute in **R** is non-transitively dependent on every key of **R**)

- [Carlo Zaniolo, 1982] Alternately, **R** is in 3NF iff for each of its functional dependencies $X \rightarrow A$, atleast one of the following conditions hold:
  - X contains A (that is, A is a subset of X, meaning $X \rightarrow A$ is a trivial functional dependency), or
  - X is superkey, or
  - Every element of $A - X$, the set difference between A and X, is a prime attribute (i.e., each attribute in $A - X$ in contained in some candidate key)

- {Simple statement} A relational schema **R** is in 3NF if for every FD $X \rightarrow A$ associated with **R** either
  - $A \subseteq X$ (i.e.,the FD is trivial), or
  - X is superkey of **R**, or
  - A is a pert of some key (not just superkey!)

Third Normal Form (3NF)

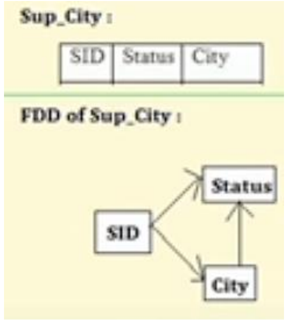
- Example of **transitive dependency**
- The functional dependency {Book} → {Author Nationality} applies; that is, if we know the book, we know the author's nationality
- Furthermore:
  - {Book} → {Author}
  - {Author} does not → {Book}
  - {Author} → {Author Nationality}
- Therefore, {Book} → {Author Nationality} is a transitive dependency
- Transitive dependency occurred because a non-key attribute (Author) was determining another non-key attribute (Author Nationality)

Book	Genre	Author	Author Nationality
Twenty Thousand Leagues Under the Sea	Science Fiction	Jules Verne	French
Journey to the Center of the Earth	Science Fiction	Jules Verne	French
Leaves of Grass	Poetry	Walt Whitman	American
Anna Karenina	Literary Fiction	Leo Tolstoy	Russian
A Confession	Religious Autobiography	Leo Tolstoy	Russian

Third Normal Form (3NF)

- **Example:**
- Sup_City(SID, Status, City) (already in 2NF)

Sup_City		
SID	**Status**	**City**
S1	30	Delhi
S2	10	Karnal
S3	40	Rohtak
S4	30	Delhi
Key: (SID)		



Partial dependencies:
SID → Status, SID → City , City → Status

Transitive dependencies:
SID → Status (as SID → City and City → Status)

Post normalization

SC	
SID	**City**
S1	Delhi
S2	Karnal
S3	Rohtak
S4	Delhi
(SID): Primary key	

CS	
City	**Status**
Delhi	30
Karnal	10
Rohtak	40
(City): Primary key	

The above two relations **SC** and **CS** are:
- Lossless join
- In 3NF
- Dependency preserving

3NF Example: Relation *dept_advisor*

- **dept_advisor**(*s_ID, i_ID, dept_name)*
  $F = \{s\_ID, dept\_name \rightarrow i\_ID,\ \ i\_ID \rightarrow dept\_name\}$

- Two candidate keys: *s_ID, dept_name,* and *i_ID, s_ID*

- *R* is in 3NF
  - $s\_ID, dept\_name \rightarrow i\_ID$ *s_ID*
    - *s_ID, dept_name* is a superkey
-
  - $i\_ID \rightarrow dept\_name$
    - *dept_name* is contained in a candidate key

- A relational schema **R** is in 3NF if for every FD $X \rightarrow A$ associated with **R** either
  - $A \subseteq X$ (i.e.,the FD is trivial), or
  - *X* is superkey of **R**, or
  - *A* is a part of some key (not just superkey!)

Redundancy in 3NF

- ***There are some redundancy in this schema***
- Example of problems due to redundancy in 3NF (*J: s_ID, K: i_ID, L: dept_name*)
- $R = (J, K, L)$
  - $F = \{JK \rightarrow L, L \rightarrow K\}$
  - And an instance table:

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

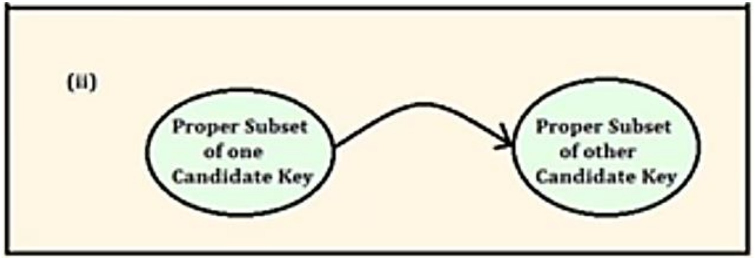
- What is wrong with the table?
  - Repetition of information (e.g., the relationship l_1, k_1)
    - (*i_ID, dept_name*)
  - Need to use null values (e.g., to represent the relationship l_2, k_2, where there is no corresponding value for J)
    - (*i_ID, dept_name*) if there is no separate relation mapping *instructors* to *departments*

Third Normal Form (3NF): Possible Redundancy

- A table is automatically in 3NF if one of the following hold:
  - If relation consists of two attributes
  - If 2NF tables consists of only one non-key attributes

- If $X \rightarrow A$ is a dependency, then the table is in the 3NF, if one of the following condition exists:
  - If X is a superkey
  - If X is a part of superkey

- If $X \rightarrow A$ is a dependency, then the table said to be NOT in 3NF, if the following holds:
  - If X is a proper subset of some key (partial dependency)
  - If X is not a proper subset of key (non key)



Third Normal Form: Motivation

- There are some situations where
  - BCNF is not dependency preserving, and
  - Efficient checking for FD violation on updates is important
- Solution: Define a weaker normal form, called Third Normal Form
  - Allows some redundancy (with resultant problems; we will see examples later)
  - But FDs can be checked on individual relations without computing a join
  - ***There is always a lossless-join, dependency-preserving decomposition into 3NF***

Testing for 3NF

- Optimization: Need to check only FDs in *F*, need not check all FDs in F^+
- Use attribute closure to check for each dependency $\alpha \rightarrow \beta$, if α is a superkey
- If α is not a superkey, we have to verify if each attribute in β is contained in a candidate key of *R*
  - This test is rather more expensive, since it involve finding candidate keys
  - Testing for 3NF has been shown to be NP-hard
  - Interestingly, decomposition into third normal form (described shortly) can be done in polynomial time

3NF Decomposition Algorithm

- Given: Relation **R**, set ***F*** of functional dependencies
- Find: Decomposition of **R** into a set of 3NF relation $\mathbf{R_i}$
- Algorithm:
  - Eliminate redundant FDs, resulting in a canonical cover F_c of F
  - Create relation $\mathbf{R_i}$ = **XY** for each FD $X \rightarrow Y$ in F_c
  - If the key of **R** does not occur in any relation $\mathbf{R_i}$, create one more relation $\mathbf{R_i}$ = **K**

- The algorithm (in next slide) ensures:
  - Each relation schema R_i is in 3NF
  - Decomposition is dependency preserving and lossless-join

3NF Decomposition Algorithm

Let F_c be a canonical cover for *F;*
i := 0;
for each functional dependency $\alpha \rightarrow \beta$ in F_c **do**
    if none of the schemas R_j, $1 \leq j \leq i$ contains $\alpha\ \beta$
        then begin
            $i := i + 1$;
            $R_i := \alpha\ \beta$
        end
if none of the schemas R_j, $1 \leq j \leq i$ contains a candidate key for R
        then begin
            $i := i + 1$;
            R_i := any candidate key for *R;*
        end
/* 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-1;
return $(R_1, R_2, ..., R_i)$

3NF Decomposition Algorithm

- Relation schema:

  cust_banker_branch = (<u>*customer_id, employee_id*</u>, *branch_name, type*)

- The functional dependencies for this relation schema are:

  $$customer\_id,\ employee\_id \rightarrow branch\_name,\ type$$
  $$employee\_id \rightarrow branch\_name$$
  $$customer\_id,\ branch\_name \rightarrow employee\_id$$

- We first compute a canonical cover
  - *branch_name* is extraneous in the r.h.s. of the 1st dependency
  - No other attribute is extraneous, so we get F_C =

  $$customer\_id,\ employee\_id \rightarrow type$$
  $$employee\_id \rightarrow branch\_name$$
  $$customer\_id,\ branch\_name \rightarrow employee\_id$$

3NF Decomposition Algorithm

- The **for** loop generates following 3NF schema:

  (*customer_id, employee_id, type*)

  (*employee_id*, *branch_name*)

  (*customer_id, branch_name, employee_id)*

  – Observe that (*customer_id, employee_id, type*) contains a candidate key of the original schema, so no further relation schema needs be added

- At end of for loop, detect and delete schemas, such as (*employee_id*, *branch_name*), which are subsets of other schemas
  – Result will not depend on the order in which FDs are considered

- The resultant simplified 3NF schema is:

  (*customer_id, employee_id, type*)

  (*customer_id, branch_name, employee_id)*

Testing for BCNF

- To check if a non-trivial dependency $\alpha \rightarrow \beta$ causes a violation of BCNF
  - 1. Compute α^+ (the attribute closure of α), and
  - 2. Verify that it includes all attributes of R, that is, it is a superkey of R

- **Simplified test**: To check if a relation schema R is in BCNF, it suffices to check only the dependencies in the given set F for violation of BCNF, rather than checking all dependencies in F^+
  - If none of the dependencies in F causes a violation of BCNF, then none of the dependencies in F^+ will cause a violation of BCNF either

- However, **simplified test** using only F is incorrect when testing a relation in a decomposition of R
  - Consider $R = (A, B, C, D, E)$, with $F = \{A \rightarrow B, BC \rightarrow D\}$
    - Decompose R into $R_1 = (A, B)$ and $R_2 = (A, C, D, E)$
    - Neither of the dependencies in F contain only attributes from (A, C, D, E) so we might be mislead into thinking R_2 satisfies BCNF
    - In fact, dependency $AC \rightarrow D$ in F^+ shows R_2 is not in BCNF

Testing Decomposition for BCNF

To check if a relation $\mathbf{R_i}$ in a decomposition of **R** is in BCNF

- Either test $\mathbf{R_i}$ for BCNF with respect to the **restriction** of F^+ to R_i (that is, all FDs in F^+ that contain only attributes from R_i)
- Or, use the original set of dependencies *F* that hold on *R*, but with the following test:
  - For every set of attributes $\alpha \subseteq R_i$, check that α^+ (the attribute closure of α) either includes no attribute of $R_i - \alpha$, or includes all attributes of R_i
  - If the condition is violated by some $\alpha \rightarrow \beta$ in F^+, the dependency

    $$\alpha \rightarrow (\alpha^+ - \alpha) \cap R_i$$

    can be shown to hold on R_i, and R_i violates BCNF
  - We use above dependency to decompose R_i

BCNF Decomposition Algorithm

- For all dependencies $A \rightarrow B$ in F^{+}, check if A is a superkey
  - By using attribute closure

- If not then,
  - Choose a dependency in F^{+} that breaks the BCNF rules, say $A \rightarrow B$
  - Create $\mathbf{R_1} = (AB)$
  - Create $\mathbf{R_2} = A\ (\mathbf{R} - (B - A))$
  - Note that: $\mathbf{R_1} \cap \mathbf{R_2} = A$ and $A \rightarrow AB$ (= $\mathbf{R_1}$), so this is a lossless decomposition

- Repeat for F_1^{+} to be all dependencies in F that contain only attributes in $\mathbf{R_1}$
  - Similarly F_2^{+}

BCNF Decomposition Algorithm

result := {*R* };
done := false;
compute F^+;
while (not *done*) **do**
    if (there is a schema R_i in *result* that is not in BCNF)
    then begin
        let $\alpha \rightarrow \beta$ be a nontrivial functional dependency that holds on R_i such that $\alpha \rightarrow R_i$ is not in F^+,
        and $\alpha \cap \beta = \emptyset$;
        $result := (result - R_i) \cup (R_i - \beta) \cup (\alpha, \beta);$
    end
    else *done* := **true;**

Note: Each R_i is in BCNF, and decomposition is lossless-join

Example of BCNF Decomposition

- **R**(A, B, C)

- Functional dependencies:
  - $A \rightarrow B$
  - $B \rightarrow C$

- Key = {A}

- **R** is not in BCNF ($B \rightarrow C$ but B is nor superkey)

- BCNF Decomposition:
  - $\mathbf{R_1}$ = (B, C)
  - $\mathbf{R_2}$ = (A, B)

Example of BCNF Decomposition

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

- Functional dependencies:
  - *course_id*→ *title*, *dept_name*, *credits*
  - *building*, *room_number*→*capacity*
  - *course_id*, *sec_id*, *semester*, *year*→*building*, *room_number*, *time_slot_id*

- A candidate key {*course_id*, *sec_id*, *semester*, *year*}

- BCNF Decomposition:

  course_id→ *title*, *dept_name*, *credits* holds

  - But *course_id* is not a superkey
  - We replace *class* by:

    course(*course_id*, *title*, *dept_name*, *credits*)

    class-1 (*course_id*, *sec_id*, *semester*, *year*, *building*, *room_number, capacity*, *time_slot_id*)

BCNF Decomposition

- *course* is in BCNF
  - How do we know this?

- *building*, *room_number*→*capacity* holds on *class-1*
  - But {*building*, *room_number*} is not a superkey for *class-1*.
  - We replace *class-1* by:
    - *classroom* (*building*, *room_number*, *capacity*)
    - *section* (*course_id*, *sec_id*, *semester*, *year*, *building*, *room_number*, *time_slot_id*)

- *classroom* and *section* are in BCNF

BCNF and Dependency Preservation

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

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

- R is not in BCNF

- Any decomposition of R will fail to preserve:

$$JK \rightarrow L$$

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

Comparison of BCNF and 3NF

- It is always possible to decompose a relation into a set of relations that are in 3NF such that:
  - The decomposition is lossless
  - The dependencies are preserved

- It is always possible to decompose a relation into a set of relations that are in BCNF such that:
  - The decomposition is lossless
  - It may not be possible to preserve dependencies

S#	3NF	BCNF
1.	It concentrates on *primary key*	It concentrates on *candidate key*
2.	Redundancy is high as compared to BCNF	0% redundancy
3.	It may preserve all the dependencies	It may not preserve all the dependencies
4.	A dependency $X \rightarrow Y$ is allowed in 3NF if X is a *superkey* or Y is a part of some key	A dependency $X \rightarrow Y$ is allowed if X is a *superkey*

Comparison of BCNF and 3NF

- Advantages to 3NF over BCNF
  - It is always possible to obtain a 3NF design without sacrificing losslessness or dependency preservation
- Disadvantages to 3NF
  - We may have to use null values to represent some of the possible meaningful relationships among data items
  - There is the problem of repetition of information

Next Lecture

Normalization

Thank you for your attention...

Any question?

Contact:
Department of Information Technology, NITK Surathkal, India
6th Floor, Room: 13
Phone: +91-9477678768
E-mail: shrutilipi@nitk.edu.in