DATABASE MANAGEMENT SYSTEM – MCA
III Semester
AUG-2022

Relational Database Design

- Features of Good Relational Design
- Atomic Domains and First Normal Form
- Decomposition Using Functional Dependencies
- Functional Dependency Theory
- Algorithms for Functional Dependencies
- More Normal Form
- Database-Design Process
- Modeling Temporal Data

Features of a Good relational database design

- Minimum **Redundancy**
  - Storage of same data in more than one location - **redundancy**
  - Disadvantage: Wastage of storage space
  - Results in updation anomalies
- Fewer **null values** in tuples
  - Values of all the attributes of a tuple are not known, so null value is to be stored
  - Cannot be provided to primary key
  - Wastage of storage space
  - Interpreting them in aggregate functions become difficult

Update Anomalies

- **Insertion anomaly:** It leads to a situation in which certain information cannot be inserted into a relation unless some other information is stored.
- **Deletion anomaly:** It leads to a situation in which deletion of data representing certain information results in losing data representing some other information that is associated with it
- **Modification Anomaly:** It leads to a situation in which repeated data changed at one place results in inconsistency unless the same data are also changed at other places

Example: Insert, Delete & Update Anomalies

StudentNum	CourseNum	Student Name	Address	Course
S21	9201	Jones	Edinburgh	Accounts
S21	9267	Jones	Edinburgh	Accounts
S24	9267	Smith	Glasgow	physics
S30	9201	Richards	Manchester	Computing
S30	9322	Richards	Manchester	Maths

INSERT: Can not insert A course if No student Opted that Course

DELETE: Assume S24 is the only one student who opted Physics. When S24 leaves the course then Student record is deleted which also result in loss of Course information.

UPDATE: If S21 changes his ***Address*** then in both records ***Address*** is to be modified otherwise it results into inconsistency.

Combine Schemas?

- Suppose we combine *instructor* and *department* into ***inst_dept***
  - Inst_dept(ID,Name,Salary,Dept_Name,Building,Budget)
  - *(Do not misunderstand as relationship set **inst_dept**)*
- Result is possible **repetition of information**

ID	*name*	*salary*	*dept_name*	*building*	*budget*
22222	Einstein	95000	Physics	Watson	70000
12121	Wu	90000	Finance	Painter	120000
32343	El Said	60000	History	Painter	50000
45565	Katz	75000	Comp. Sci.	Taylor	100000
98345	Kim	80000	Elec. Eng.	Taylor	85000
76766	Crick	72000	Biology	Watson	90000
10101	Srinivasan	65000	Comp. Sci.	Taylor	100000
58583	Califieri	62000	History	Painter	50000
83821	Brandt	92000	Comp. Sci.	Taylor	100000
15151	Mozart	40000	Music	Packard	80000
33456	Gold	87000	Physics	Watson	70000
76543	Singh	80000	Finance	Painter	120000

Redundancy

A Combined Schema Without Repetition

- Consider combining relations
  - ***sec_class (sec_id, building, room_number)*** and
  - ***Section (course_id, sec_id, semester, year)***

  into one relation
  - ***Section (course_id, sec_id, semester, year, building, room_number)***

- No repetition in this case - This is a Special case. It is not TRUE always.

- Combining Relation may result into Redundancy.

What About Smaller Schemas?

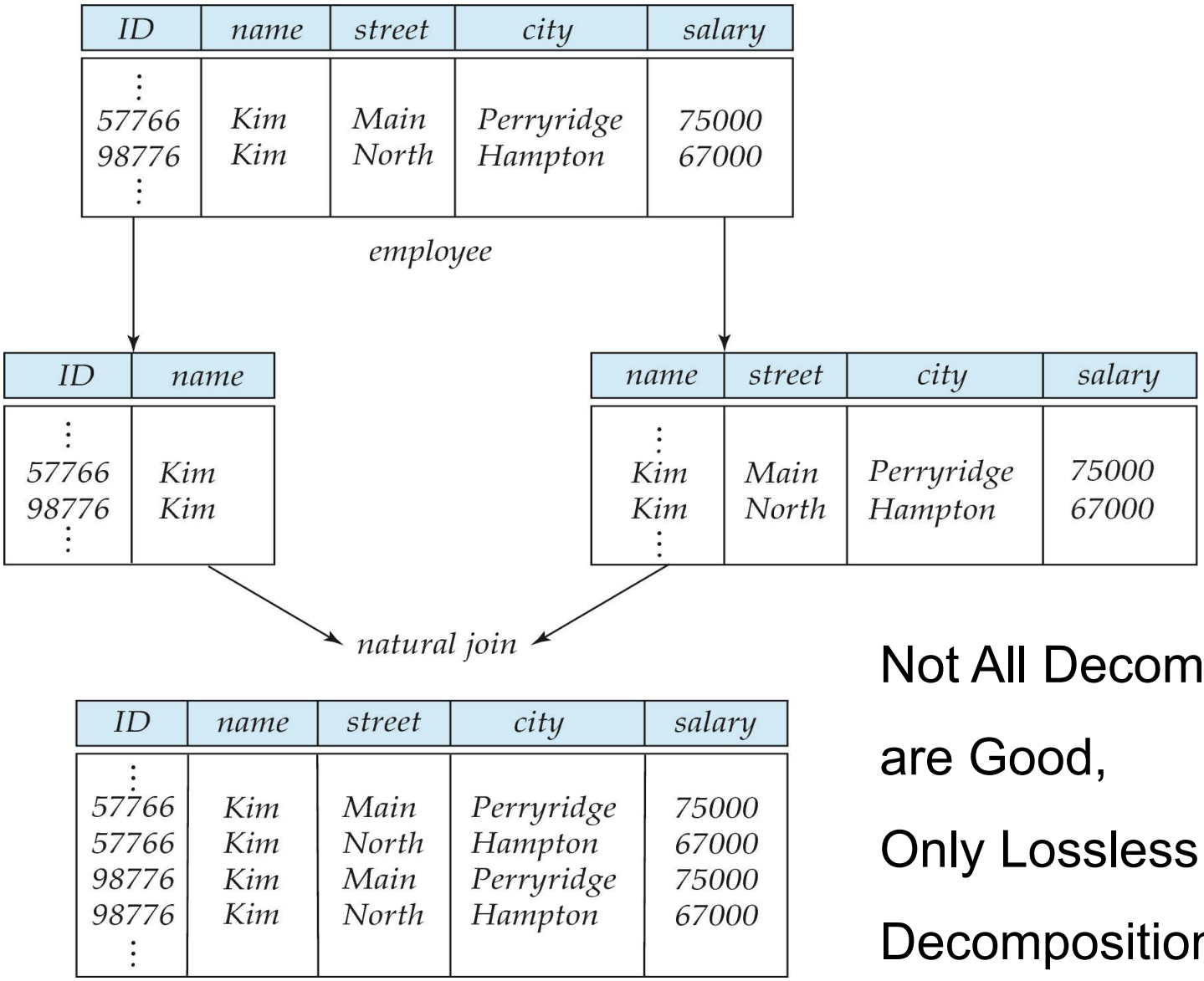
- Suppose we had started with ***inst_dept.*** How would we know to split up (**decompose**) it into ***instructor*** **and** ***department***?
  - **Not all decompositions are good**. Suppose we decompose

    employee(ID, name, street, city, salary) into

    employee1 (*ID*, *name*) and

    employee2 (*name*, *street, city, salary*)
- The next slide shows how we lose information -- we cannot reconstruct the original *employee* relation -- and so, this is a **lossy decomposition**.

A Lossy Decomposition



Not All Decompositions are Good,

Only Lossless Decomposition is Good

Example of Lossless-Join Decomposition

- **Lossless join decomposition**
- Decomposition of *R* = *(A, B, C)*

  $R_1 = (A, B)$ $R_2 = (B, C)$

A	B	C
α	1	A
β	2	B

r

A	B
α	1
β	2

$\Pi_{A,B}(r)$

B	C
1	A
2	B

$\Pi_{B,C}(r)$

$\Pi_A (r) \bowtie \Pi_B (r)$

A	B	C
α	1	A
β	2	B

How to avoid redundancy & achieve Lossless Decomposition

Decomposition of a Relation

- While designing a relational database certain problems that are met due to redundancy and null values can be resolved by decomposing a relation
- In decomposing, a relation is replaced with a collection of smaller relations with specific relationship between them
- Formally, the **decomposition** of a relation schema R is defined as its replacement by a set of relation schemas such that each relation schema contains a subset of attributes of R.

Goal — Devise a Theory for the Following

- Decide whether a particular relation *R* is in "good" form.
- In the case that a relation *R* is not in "good" form, decompose it into a set of relations $\{R_1, R_2, \ldots, R_n\}$ such that
  - each relation is in **good form**
  - the decomposition is a **lossless-join decomposition**
- Our theory is based on:
  - functional dependencies

Decomposition using Functional Dependencies

- Functional dependency is a **relationship between two attributes.**
- Functional dependencies are the **special forms of integrity** constraints that **generalize the concepts of keys**
- They play a key role in developing a good database schema
- Assume that ***K*** is some attribute/s of **R**
  - **i.e. K $\subseteq$ *R***
  - we say that ***K is a super key of r* (*R*)** if the functional dependency

    K*→*R holds on *r* (*R*).

Functional Dependencies (Cont.)

- Let *R* be a relation schema

$$\alpha \subseteq R \quad and \quad \beta \subseteq R$$

- The **functional dependency**

$$\alpha \rightarrow \beta$$

holds on *R* if and only if for any legal relations *r*(R), whenever any two tuples t_1 and t_2 of *r* agree on the attributes α, they also agree on the attributes β. That is,

$$t_1[\alpha] = t_2[\alpha] \quad \Rightarrow \quad t_1[\beta] = t_2[\beta]$$

Example: Functional Dependency

R={A,B,C,D}

Let α={A} & β= {B}

α & β are subsets of R

Is A → B holds ?

Is A →C holds?

Let α={A,B} & β= {C,D}

α & β are subsets of R

Is (A,B) →(C ,D) holds?

Is (A,C) →D holds ?

A	B	C	D
a_1	b_1	c_1	d_1
a_1	b_2	c_1	d_2
a_2	b_2	c_2	d_2
a_2	b_3	c_2	d_3
a_3	b_3	c_2	d_4

Note: that Functional dependency is not determined by the data appearing in a relation at a given point of time, instead it **depends on the meaning(semantics) of the attributes.**

Functional Dependencies

If one set of attributes in a table determines another set of attributes in the table, then the second set of attributes is said to be functionally dependent on the first set of attributes.

Example 1

ISBN	Title	Price
0-321-32132-1	Balloon	$34.00
0-55-123456-9	Main Street	$22.95
0-123-45678-0	Ulysses	$34.00
1-22-233700-0	Visual Basic	$25.00

Table Scheme: {ISBN, Title, Price}
Functional Dependencies: {ISBN} → {Title}
{ISBN} → {Price}

Functional Dependencies

Example 2

PubID	PubName	PubPhone
1	Big House	999-999-9999
2	Small House	123-456-7890
3	Alpha Press	111-111-1111

Table Scheme: {PubID, PubName, PubPhone}
Functional Dependencies: {PubId} → {PubPhone}
{PubId} → {PubName}
{PubName, PubPhone} → {PubID}

Example 3

AuID	AuName	AuPhone
1	Sleepy	321-321-1111
2	Snoopy	232-234-1234
3	Grumpy	665-235-6532
4	Jones	123-333-3333
5	Smith	654-223-3455
6	Joyce	666-666-6666
7	Roman	444-444-4444

Table Scheme: {AuID, AuName, AuPhone}
Functional Dependencies: {AuId} → {AuPhone}
{AuId} → {AuName}
{AuName, AuPhone} → {AuID}

Functional Dependencies

Example 3

Assume **ITEMS(it_name, comp_name, price)** is a schema storing items(toothbrush, toothpaste etc.) and company(Colgate, Pepsodent etc.) names and price.

It_name	Comp_name	price
Toothpast	Colgate	20
Toothpast	Pepsodent	30
Toothbrush	Colgate	55
Toothbrush	Pepsodent	90

An It_name may be produced by many companies ,

therefore **It_name $\not\rightarrow$ Comp_Name**

($\not\rightarrow$ **means NOT -Functionally depends**)

A company may be producing many It_name,

therefore **Comp_Name $\not\rightarrow$ It_name**

Functional Dependencies

Example 3 (Contd.)

It_name	Comp_name	price
Toothpast	Colgate	20
Toothpast	Pepsodent	30
Toothbrush	Colgate	55
Toothbrush	Pepsodent	90

Similarly price is not functionally depends on It_name or Comp_name

It_name $\not\rightarrow$ Price & Comp_Name $\not\rightarrow$ rice

However a Price is uniquely determined by the combination of (It_Name,Comp_Name), therefore

(It_Name,Comp_Name) $\rightarrow$ Price

Trivial Functional Dependency

- *A* functional dependency is **trivial** if it is satisfied by all relations
  - $X \rightarrow X$ is satisfied by all relations having attribute X.
  - Similarly, $XY \rightarrow X$ is satisfied by all relations having attributes X and Y.
- Example*:*
  - *ID, name* $\rightarrow$ *ID*
  - *name* $\rightarrow$ *name*

In general, $\alpha \rightarrow \beta$ is trivial if $\beta \subseteq \alpha$

- The dependencies that are not trivial are called **non-trivial dependencies.**
- These are the dependencies that correspond to the essential integrity constraints.

Functional Dependencies
Super Key & Candidate key

- K is a super key for relation schema R if and only if $K \rightarrow R$
- K is a candidate key (minimal Super Key) for R if and only if
  - $K \rightarrow R$ *(means K is Super key)*, and
  - for **no** $\alpha \subset K$, $\alpha \rightarrow R$

Example: Minimal Super Key

- ***Consider the ITEMS relation.***

Let us take K= **(It_Name,Comp_Name)**

A Price is uniquely determined by the combination of (It_Name,Comp_Name), therefore

(It_Name,Comp_Name) ⟶ Price

Following two functional dependency also holds because of trivial functional dependency (i.e. $\alpha \rightarrow \beta$ holds always if $\beta \subset \alpha$)

(It_Name,Comp_Name) → It_Name **Trivial**

because It_Name ⊂ (It_Name,Comp_Name)

(It_Name,Comp_Name) → Comp_Name

because Comp_Name⊂ (It_Name,Comp_Name)

Therefore (It_Name,Comp_Name) ⟶ ITEMS

(It_Name,Comp_Name) is Super Key for ITEMS

Is it Minimal Super key also?

Example: Minimal Super Key (Contd.)

As Discussed in previous slide **(It_name, Comp_name) is Super Key.**

- Further **K** i.e. **(It_name, Comp_name)** can be a **candidate key** (means minimal Super key) ,

  if **none of the subset of K *determine* R** i.e. **ITEMS.**

"None of the sub sets of (It_name, Comp_name) is Super key"

Subsets of (It_name, Comp_name) are

(It_name) & (Comp_Name)

We know **It_name alone do not determine ITEMS & Comp_Name alone also do not determine ITEMS .**

i.e. Neither It_name nor Comp_Name is Super Key for ITEMS

- **i.e It_name $\not\rightarrow$ ITEMS & Comp_name $\not\rightarrow$ ITEMS**

Example: Minimal Super Key (Contd.)

- From Super key, **(It_name, Comp_name)** we are unable to find any **subset** which **determine ITEMS** .

not possible to find any sub set of

(**It_name, Comp_name) which is super key**

- i.e. Hence **(It_name, Comp_name)** are **minimum attributes required** to determine ITEMS.

- Therefore **(It_name, Comp_name) is minimal Super key (i.e. candidate key)**

Functional Dependencies(Cont'd)

- Salesperson(Salesperson number, Salesperson name, Commission Percentage, Year of Hire, Department Number, Manager Name)
- Product(Product Number, Product Name, Unit Price, Quantity)

Salesperson Number → Salesperson Name
Salesperson Number → Commission Percentage
Salesperson Number → Year of Hire
Salesperson Number → Department Number
Salesperson Number → Manager Name
Product Number → Product Name
Product Number → Unit Price
Department Number → Manager Name
Salesperson Number, Product Number → Quantity

Write the Functional Dependencies for a given System Requirements

- Assume that we want to store information about employees and department in which they are working.
- Every employee has a unique EmpNo. About each employee we are interested to store his name ,salary what he earns, city in which he resides , pincode , STDCode of the city and PhoneNumber of the employee. Every city has one pincode (ignoring the point that-with in a city different areas will have different Pincode) , similar type assumption we considered here like every city has one STDCode.
- Each employee works exactly in one department. More than one employees work in a department. About each department we want to record department number-Deptno and department name Dname, Department location –Loc. Every department has a unique deptno, every department has one unique Dname and each department is situated in one Location.

See sample data corresponding to these constraints in next slide

Sample data corresponding to the constraints in previous slide

Empno	Ename	Sal	City	PinCode	STD_Code	Phone_Number	Deptno	Dname	Loc
157	Niraj	1000	MANIPAL	576104	820	12345	D1	ADMINSTION	MANGALORE
100	Ravi	2000	MANGALORE	587693	819	56788	D2	MARKETING	BOMBAY
102	Raviraj	3899	BANGLAORE	580001	80	786536	D3	RESEARCH	BOMBAY
111	Raghu	5788	MANIPAL	576104	820	125978	D1	ADMINSTION	MANGALORE
150	X	2467	BOMBAY	489921	420	678899	D2	MARKETING	BOMBAY
103	Y	9763	HYDERBAD	765690	560	567899	D3	RESEARCH	BOMBAY
109	Z	46789	CHENNAI	674321	730	656890	D1	ADMINSTION	MANGALORE
125	Manu	5788	BANGLAORE	580001	80	123456	D3	RESEARCH	BOMBAY
104	A	8653	MANIPAL	576104	820	89865	D2	MARKETING	BOMBAY
106	B	79076	CHENNAI	674321	730	566778	D3	RESEARCH	BOMBAY
123	Mahesh	2000	MANGALORE	587693	819	55677	D2	MARKETING	BOMBAY
108	Ravi	44466	CHENNAI	674321	730	57780	D4	PRODUCTION	MANGALORE
124	D	44467	HYDERBAD	765690	560	786535	D5	PURCHASE	CHENNAI
113	Z	65576	MANIPAL	576104	820	89627	D5	PURCHASE	CHENNAI

Set of Functional Dependencies that hold

Following are the some FDs that hold on the system as per the requirements given in slide 26

Empno → Name; Empno → Sal ; Empno → Deptno;

Empno → Dname; Empno → City; Empno → **PinCode;**

Empno → **STD_Code;** ***Empno →*** **Phone_Number;** ***Empno →*** **Loc**

Deptno→ Dname; Deptno→ Loc; Dname → Deptno;

Dname → Loc; City →PinCode; PinCode → City;

City → STD_Code; STD_Code → ***City***

Whether these Functional Dependency Hold ?

Deptno → Ename; Deptno → Sal; Dname → Sal;

City → Dname; City →Deptno; Phone_Number → STD_Code;

City→PhoneNumber ; **PinCode →STD_Code**

IS (Empno , Name) → Deptno ;

Functional Dependencies(Cont'd)

Full Functional dependency:

- If A and B are attributes of a table, B is fully functionally dependent on A if B is functionally dependent on A, but not on any proper subset of A.
- **Example**
  Consider the example **ITEMS(It_Name, Comp_Name, Price)**
- Consider Functional Dependency **(It_Name, Comp_Name) → Price**
- **Price** is **Fully Functionally dependent** because-

- It_Name $\not\rightarrow$ Price
- Comp_Name) $\not\rightarrow$ Price

} Price do not Functionally dependent on none of the proper subset of (It_Name, Comp_Name)

Functional Dependencies(Cont'd)

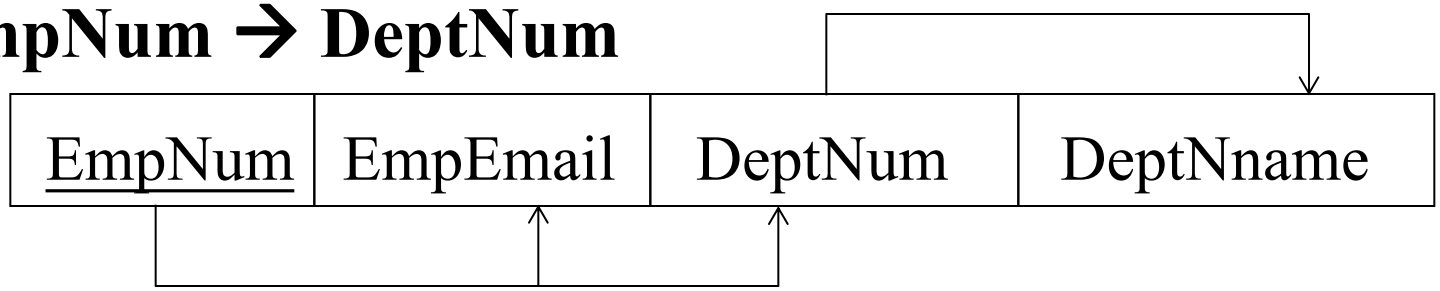
Partial Functional Dependency:

- If A and B are attributes of a table, B is partially dependent on A if there is some attribute that can be removed from A and yet the dependency still holds.
- **Example**

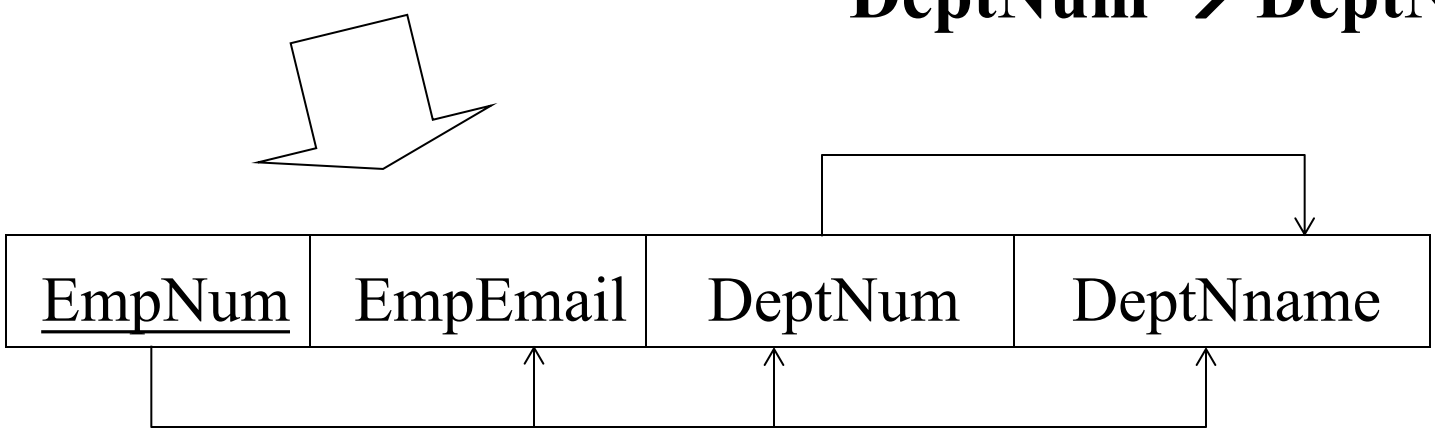
  (Empno, Name) → Deptno , but ***Empno → Deptno***
- Deptno is also dependent on Empno which is subset of (Empno,Name)
- Therefore Deptno is not fully dependent on (Empno,Name),
- Means **Deptno** is **Partially Dependent** on **(Empno, Name)**

Transitive dependency

EmpNum → DeptNum



DeptNum → DeptName



DeptName is *transitively dependent* on EmpNum via DeptNum

EmpNum → DeptName

Use of Functional Dependencies

- We **use functional dependencies** to:
  - **test relations** to see if they are legal under a given set of functional dependencies *F*.
    - If a relation *r* is legal under a set *F* of functional dependencies, we say that *r* **satisfies** *F.*
  - **specify constraints** on the set of legal relations*
    - We say that *F* **holds on** *R* if all legal relations on *R* satisfy the set of functional dependencies *F.*
- Note: A specific instance of a relation schema may satisfy a functional dependency even if the functional dependency does not hold on all legal instances.
  - For example, a specific instance of *instructor* may, by chance, satisfy

    name $\rightarrow$ *ID.*

**** See the notes section***

Normalization

- Normalization is the process of modifying a relation schema based on its Functional Dependencies and primary keys so that it meets certain rules called normal forms.
- It is conducted by evaluating a relation schema to check whether it satisfies particular rules and if not, then decomposing the schema into set of smaller relation schemas that do not violate those rules
- Normalization can be considered as fundamental to the modelling and design of a relational database
- Main purpose is to eliminate data redundancy and avoid data update anomalies

Normalization

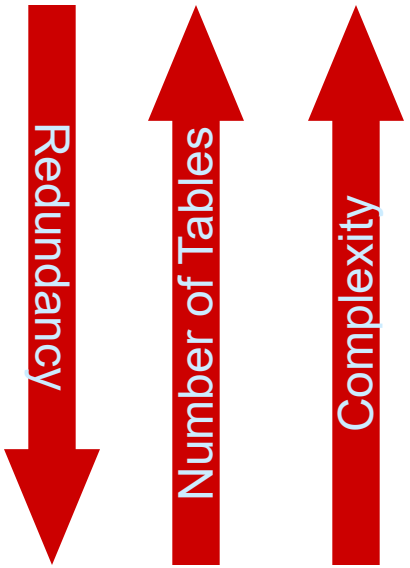
- A relation is said to be in a particular normal form if it satisfies certain specified constraints
- Each of the normal form is stricter than its predecessors
- The normal forms are used to ensure that various types of anomalies and inconsistencies are removed from the database

Normalization

- A properly normalized database should have the following characteristics
  - Atomic values in each fields
  - Absence of redundancy.
  - Minimal use of null values.
  - Minimal loss of information.

Levels of Normalization

- Levels of normalization based on the amount of redundancy in the database.
- Various levels of normalization are:
  - First Normal Form (1NF)
  - Second Normal Form (2NF)
  - Third Normal Form (3NF)
  - Boyce-Codd Normal Form (BCNF)
  - Fourth Normal Form (4NF)
  - Fifth Normal Form (5NF)
  - Domain Key Normal Form (DKNF)



Most databases should be 3NF or BCNF in order to avoid the database anomalies.

First Normal Form

- A relational schema R is in **first normal form** if the domains of all attributes of R are atomic
- Domain is **atomic** if its elements are considered to be indivisible units
  - Examples of **non-atomic** domains:
    - Set of names, composite attributes
    - Identification numbers like "CS101" that can be broken up into parts-Dept & RollNo
- **Non-atomic** values complicate storage and encourage redundant (repeated) storage of data
  - Example: Set of accounts stored with each customer, and set of owners stored with each account

First Normal Form (Cont'd)

- Atomicity is actually a property of how the elements of the domain are used.
  - Example: Strings would normally be considered indivisible
  - Suppose that students are given roll numbers which are strings of the form *CS0012* or *EE1127*
    - If the first two characters are extracted to find the department, the domain of roll numbers is not atomic.
  - Doing so is a bad idea: leads to encoding of information in application program rather than in the database.

Example

TABLE_PRODUCT

Product ID	Color	Price
1	red, green	15.99
2	yellow	23.99
3	green	17.50
4	yellow, blue	9.99
5	red	29.99

1NF Decomposition

1. Place all items that appear in the repeating group in a new table
2. Designate a primary key for each new table produced.
3. Duplicate in the new table the primary key of the table from which the repeating group was extracted or vice versa.

1NF Decomposition

TABLE_PRODUCT_PRICE

Product ID	Price
1	15.99
2	23.99
3	17.50
4	9.99
5	29.99

TABLE_PRODUCT_COLOR

Product ID	Color
1	red
1	green
2	yellow
3	green
4	yellow
4	blue
5	red

Recap

- Functional Dependency
- Super key
- Candidate Key
- Trivial Fd & Non-Trivial Fd
- Full functional Dependency
- Partial functional Dependency
- Normal Forms
- 1st Normal Form

Closure of a Set of Functional Dependencies

- Given a set ***F*** of functional dependencies, there are certain **other functional dependencies** that are **logically implied by *F*.**
  - For example: If $A \rightarrow B$ and $B \rightarrow C$, then we can infer that $A \rightarrow C$
- The set of **all** functional dependencies logically implied by *F* is the **closure** of *F*.
- We denote the ***closure* of *F*** by $\mathbf{F^+}$.
- $\mathbf{F^+}$ is a superset of ***F***. $\mathbf{F^+} \supset \boldsymbol{F}$

Set of Functional Dependencies that hold

Following are the some FDs that hold on the system as per the requirements given in slide 26

Empno → Name; Empno → Sal ; Empno → Deptno;
Empno → Dname; Empno → City; Empno → PinCode;
Empno → STD_Code; Empno →Phone_Number; Empno →Loc
Deptno → Dname; Deptno → Loc; Dname → Deptno;
Dname → Loc; City →PinCode; PinCode → City;
City → STD_Code; STD_Code → City

Whether these Functional Dependency Hold ?

Deptno → Ename; Deptno → Sal; Dname → Sal;
City → Dname; City →Deptno; Phone_Number → STD_Code;
City→PhoneNumber ; PinCode →STD_Code
IS (Empno , Name) → Deptno ;

Closure of a Set of Functional Dependencies

- We can find F^+, the closure of F, by repeatedly applying **Armstrong's Axioms:**
  - if $\beta \subseteq \alpha$, then $\alpha \rightarrow \beta$ **(reflexivity)**
  - if $\alpha \rightarrow \beta$, then $\gamma\ \alpha \rightarrow \gamma\ \beta$ **(augmentation)**
  - if $\alpha \rightarrow \beta$, and $\beta \rightarrow \gamma$, then $\alpha \rightarrow \gamma$ **(transitivity)**
- These rules are
  - **sound** (generate only functional dependencies that actually hold), and
  - **complete** (generate all functional dependencies that hold).

Closure of Functional Dependencies (Cont.)

- Additional rules:
  - If $\alpha \rightarrow \beta$ holds *a*nd $\alpha \rightarrow \gamma$ holds, then $\alpha \rightarrow \beta\gamma$ holds **(union)**
  - If $\alpha \rightarrow \beta\,\gamma$ holds, then $\alpha \rightarrow \beta$ holds and $\alpha \rightarrow \gamma$ holds **(decomposition)**
  - If $\alpha \rightarrow \beta$ holds *a*nd $\gamma\ \beta \rightarrow \delta$ holds, then $\alpha\ \gamma \rightarrow \delta$ holds **(pseudotransitivity)**

  The above rules can be inferred from Armstrong's axioms.

* For Proof, see notes section.

Example: **Finding some logically implied functional dependencies using Armstrong Axioms**

- $R = (A, B, C, G, H, I)$

  $F = \{ A \rightarrow B$
  $A \rightarrow C$
  $CG \rightarrow H$
  $CG \rightarrow I$
  $B \rightarrow H \}$

if $\beta \subseteq \alpha$, **then** $\alpha \rightarrow \beta$
(reflexivity)
if $\alpha \rightarrow \beta$, **then** $\gamma\ \alpha \rightarrow \gamma\ \beta$
(augmentation)
if $\alpha \rightarrow \beta$, **and** $\beta \rightarrow \gamma$, **then** $\alpha \rightarrow \gamma$
(transitivity)

- some members of F^+
  - $A \rightarrow H$
    - by transitivity from $A \rightarrow B$ *and* $B \rightarrow H$
  - $AG \rightarrow I$
    - Since $A \rightarrow C$ and $CG \rightarrow I$ *(pseudotransitivity)*
  - $CG \rightarrow HI$
    - *Since* $CG \rightarrow H$ *and* $CG \rightarrow$ I (Union rule)

Procedure for Computing F^+

- To compute the closure of a set of functional dependencies **F**:

$F^+ = F$

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

      then add the resulting functional dependency to F^+

until F^+ does not change any further

Closure of Attribute Sets

- Given a set of attributes **a**, define the ***closure*** of a **under *F*** (denoted by a^+) as the set of attributes that are functionally determined by a under ***F***

Algorithm to **compute a^+, the closure of a** under ***F***

```
result := a;
while (changes to result) do
    for each β → γ in F do
      begin
        if β ⊆ result then result := result ∪ γ
      end
```

Example for Attribute Set Closure

- $R = (A, B, C, G, H, I)$
- $F = \{ A \rightarrow B$
  $A \rightarrow C$
  $CG \rightarrow H$
  $CG \rightarrow I$
  $B \rightarrow H \}$ Find AG^+
- $(AG)^+$
  1. *result* = AG
  2. *result* = $ABCG$ ($A \rightarrow C$ and $A \rightarrow B$)
  3. *result* = $ABCGH$ ($CG \rightarrow H$ and $CG \subseteq AGBC$)
  4. *result* = $ABCGHI$ ($CG \rightarrow I$ and $CG \subseteq AGBCH$)

Algorithm to **compute a^+**, the **closure** of **a** under ***F***

```
result := a;
while (changes to result) do
    for each β → γ in F do
        begin
            if β ⊆ result then result := result ∪ γ
        end
```

Step-by-step approach to Find AG^+

1. *result = AG*
2. *While loop* (*1st While loop*)
3. *For Loop*

1. take $A \rightarrow B$

$A \subseteq$ result

result = result $\cup$ *B* ={ *AGB*}

2. take $A \rightarrow C$

$A \subseteq$ result

result = result $\cup$ *C* ={ *AGBC*}

3. Take $CG \rightarrow H$

$CG \subseteq$ result

result = result $\cup$ *H* ={ *AGBCH*}

4. Take $CG \rightarrow I$

$CG \subseteq$ result

result = result $\cup$ *I* ={ *AGBCHI*}

5. Take $B \rightarrow H$

$B \subseteq$ result

result = result $\cup$ *H* ={ *AGBCHI*}

End of For Loop

….Step-by-step approach to Find AG^+

At the end of 1st While loop result= { AGBCHI}
So when 2nd While loop starts result= { AGBCHI}

1. *While loop (2nd While loop)*
2. *For Loop*
   1. take $A \rightarrow B$
      $A \subseteq$ result
      result = result $\cup$ *B ==*{ AGBCHI}
   2. take $A \rightarrow C$
      $A \subseteq$ result
      result = result $\cup$ *C ==*{ AGBCHI}
   3. Take $CG \rightarrow H$
      $CG \subseteq$ result
      result = result $\cup$ *H =*{ AGBCHI}
   4. Take $CG \rightarrow I$
      $CG \subseteq$ result
      result = result $\cup$ *I =*{ AGBCHI}
   5. *Take* $B \rightarrow H$
      $B \subseteq$ result
      result = result $\cup$ *H =*{ AGBCHI}

   End of For Loop

Result from While loop 1 and While loop 2 are same** , there is **no change in result** , therefore **exit While loop

Exercises

1. *R (A, B, C, D)*

   $F = \{ A \rightarrow B \ , B \rightarrow C \ , \quad B \rightarrow D \ \}$, Find A^+ , B^+

2. *R (A,B,C,D)*

   $F = \{C \rightarrow D \ , A \rightarrow B \ , B \rightarrow C \}$ Find A^+ , C^+

1.

$A^+ = \{A, B, C, D\}$

$B^+ = \{ B, C, D \}$

2.

$A^+ = \{A, B, C, D\}$

$C^+ = \{C, D \}$

Uses of Attribute Closure

There are several uses of the attribute closure algorithm:

1. **Testing for superkey:**
   - To test if α is a superkey, we compute α^+, and check if α^+ contains all attributes of *R*. i.e $\alpha \rightarrow \mathbf{R}$
2. **Testing functional dependencies**
   - To check if a functional dependency $\alpha \rightarrow \beta$ holds (or, in other words, is it in F^+), just **check if $\beta \subseteq \alpha^+$.**
   - That is, we compute α^+ by using attribute closure, and then check if it contains β.
   - Is a simple and cheap test, and very useful
3. **Computing closure of F i.e. F^+**
   - For each $\gamma \subseteq \boldsymbol{R}$, we find the closure γ^+, and for each $\boldsymbol{S} \subseteq \gamma^+$, we output a functional dependency $\gamma \rightarrow \boldsymbol{S}$.

1. Example Testing for superkey & Candidate Key:

- $R = (A, B, C, G, H, I)$
- $F = \{ A \rightarrow B$
  $A \rightarrow C$
  $CG \rightarrow H$
  $CG \rightarrow I$
  $B \rightarrow H \ \}$
- **Is AG** a Super Key ? *
- **Is *AG*** a candidate key**?** *
  1. Is AG a super key?
     1. Does $AG \rightarrow R$ *?* == Is $(AG)^+ \supseteq R$
  2. Is any subset of AG a super key?
     1. Does $A \rightarrow R$? == Is $(A)^+$ contains R
     2. Does $G \rightarrow R$? == Is $(G)^+$ contains R
- Is **A** is super key?

** See **slide 21** for testing – is the super key is also candidate key ?*

Exercise

- $R = (A, B, C, G, H, I)$
- $F = \{ A \rightarrow B$
  $A \rightarrow C$
  $CG \rightarrow H$
  $CG \rightarrow I$
  $B \rightarrow H$
  $C \rightarrow G$
- $\}$
- **Is AB** a Super Key ? *
- **Is *AB*** a candidate key**?** *

Step-by-step approach to Check AG is Super Key

Is AG is Super Key?

- Find **AG^+**
- From slide 50 we know AG^+ is **$AG^+ = \{ABCGHI\}$**
- i.e. $R \subseteq (AG)^+$
  - Means R ={A,B,C,G,H,I }, all attributes of R Contained in **AG^+**
- Therefore **$AG \rightarrow R$**
- Therefore **AG is Super key**

***Reference**: slide 21*

- *K* is a super key for relation schema *R* if and only if $K \rightarrow R$
- *K* is a candidate key (minimal Super Key) for *R* if and only if
  - $K \rightarrow R$ *(means K is Super key)*, and
  - for no $\alpha \subset K$, $\alpha \rightarrow R$

Step-by-step approach to Check AG is Candidate Key

***Reference**: slide 21*

- *K* is a super key for relation schema *R* if and only if $K \rightarrow R$
- *K* is a candidate key (minimal Super Key) for *R* if and only if
  - $K \rightarrow R$ *(means K is Super key)*, and
  - for no $\alpha \subset K$, $\alpha \rightarrow R$

IS AG Candidate key ?

We know AG is Super key.

Check further, is there any subset of AG is Super Key?

Subsets of AG are A & G

Check is $A \rightarrow R$ or $G \rightarrow R$?

If any one subset is Super Key then AG is not Candidate key, but only super key.

If none of subsets are Super key then AG is Candidate Key.

Find A^+ & G^+

$A^+ = \{ A,B,C,H \}$ **A** $\not\rightarrow$ **R** , Therefore **A** is **Not Super Key**

$G^+ = \{ G \}$ **G** $\not\rightarrow$ **R** , Therefore **G** is **Not Super Key**

None of subsets of AG are Super Keys –

Therefore **AG is Candidate Key (Minimal Super Key)**

Exercises

- Consider a Schema R = (A, B, C, D, E)
- FDs F = {A →B, B →C, C →A, AB →E, BD →A}
- IS AC →B holds ?
- IS C → E holds ?
- IS AE → D holds ?

2.Example for Testing functional dependencies

To check if a functional dependency $\alpha \rightarrow \beta$ holds (or, in other words, is it in F^+), just check if $\beta \subseteq (\alpha^+$ If β α^+ $\not\rightarrow$ n α β

R = (A, B, C, G, H, I)

$F = \{ \quad A \rightarrow B \ , A \rightarrow C \ , CG \rightarrow H \ , CG \rightarrow I \ , B \rightarrow H \ \}$

Is $CG \rightarrow A$ functional dependency holds ?

Solution:

- Find CG^+
- IF $A \in CG^+$ Then

  $CG \rightarrow A$ holds

  ELSE

  $CG \not\rightarrow A$ (means functional dependency do not hold)

Exercise:

Is $AC \rightarrow H$ holds ?	**YES**
Is $AG \rightarrow I$ holds ?	**YES**
Is $AI \rightarrow G$ holds ?	**NO**

3. Example for Computing F+

For each $\gamma \subseteq R$, we find the closure γ^+, and for each $S \subseteq \gamma^+$, we output a functional dependency $\gamma \rightarrow S$.

$R = (A, B, C)$

$F = \{ A \rightarrow B \ , B \rightarrow C \}$

All the subsets of R are –A,B,C,AB,AC,BC,ABC

Take one subset from above **set say -A** and **find A^+**

A^+ ={A ,B,C}

Subsets of A^+ *are* - A,B,C,AB,AC,BC,ABC

Therefore following functional dependencies hold

$A \rightarrow B$, $A \rightarrow C$, $A \rightarrow AB$, $A \rightarrow AC$, $A \rightarrow BC$, $A \rightarrow ABC$. ----

(1)Similarly , **take next subset say B** and **Find B^+**

B^+={B,C}

Subsets of B^+ *are* - B,C,BC

Therefore following functional dependencies hold

$B \rightarrow C$,$B \rightarrow BC$ ----(2)

Take subset C , Find C^+

C^+ ={ C}

Therefore **$C \rightarrow C$ (similarly $A \rightarrow A$ and $B \rightarrow B$ holds)** holds ----(3)

….Example for Computing F^+

Take AB

$$\mathbf{A\,B^+ = \{A, B, C\}}$$

Subsets of AB+ are - A , B , C , A B , A C , B C , A B C

Therefore following functional dependencies hold

$$\mathbf{AB \rightarrow B\ , AB \rightarrow C\ , A\,B \rightarrow AB,\ AB \rightarrow AC,\ A\,B \rightarrow BC,\ AB \rightarrow ABC}$$ ----(4)

Take BC

$$\mathbf{B\,C^+ = \{\ B, C\}}$$

Subsets of BC+ are - B , C , B C

Therefore following functional dependencies hold

$$\mathbf{BC \rightarrow B, BC \rightarrow C, BC \rightarrow BC}$$ ---- (5)

Take AC

$$\mathbf{A\,C^+ = \{A, B, C\}}$$

Subsets of AC^+ are - A , B , C , A B , A C , B C , A B C

Therefore following functional dependencies hold

$$\mathbf{AC \rightarrow B\ , AC \rightarrow C\ , AC \rightarrow AB,\ AC \rightarrow AC,\ AC \rightarrow BC,\ AC \rightarrow ABC.}$$ ----(6)

….Example for Computing F^+

Take ABC

ABC+={A , B , C }

Subsets of ABC^+ are - A , B , C , A B , A C , B C , A B C

Therefore following functional dependencies hold

ABC → B , ABC → C , ABC→ AB, ABC → AC, ABC→ BC, ABC → ABC. ----(7)

From (1) (2) (3) (4) (5) (6) (7)

by eliminating duplicates we can get all functional dependencies set

$\mathbf{F^+}$ = { $A \rightarrow A$, $B \rightarrow B$, $C \rightarrow C$, $A \rightarrow B$, $A \rightarrow C$, $A \rightarrow AB$, $A \rightarrow AC$, $A \rightarrow BC$,
$A \rightarrow ABC$, $B \rightarrow C$, $B \rightarrow BC$, $AB \rightarrow B$, $AB \rightarrow C$, $A B \rightarrow AB$, $AB \rightarrow AC$,
$A B \rightarrow BC$, $AB \rightarrow ABC$, $AC \rightarrow B$, $AC \rightarrow C$, $AC \rightarrow AB$, $AC \rightarrow AC$,
$AC \rightarrow BC$, $AC \rightarrow ABC$, $ABC \rightarrow B$, $ABC \rightarrow C$, $ABC \rightarrow AB$, $ABC \rightarrow AC$,
$ABC \rightarrow BC$, $ABC \rightarrow ABC$ }

Exercises

- Consider a relation
  - gradeInfo (rollNo, studName, course, grade)
- Suppose the following FDs hold:
  - {rollNo, course} →{grade};
  - {studName, course} →{grade} ;
  - {rollNo}→{studName};
  - {studName}→{rollNo}
- Is this a candidate Key (studName, course) ?

Canonical Cover

- Sets of functional dependencies may have redundant dependencies that can be inferred from the others
  - For example: $A \rightarrow C$ is redundant in: $\{A \rightarrow B,\ \ B \rightarrow C,\ A \rightarrow C\}$
  - Parts of a functional dependency may be redundant
    - E.g.: on RHS: $\{A \rightarrow B,\ \ B \rightarrow C,\ \ A \rightarrow CD\}$ can be simplified to

      $$\{A \rightarrow B,\ \ B \rightarrow C,\ \ A \rightarrow D\}$$

    - E.g.: on LHS: $\{A \rightarrow B,\ \ B \rightarrow C,\ \ AC \rightarrow D\}$ can be simplified to

      $$\{A \rightarrow B,\ \ B \rightarrow C,\ \ A \rightarrow D\}$$

- Intuitively, a canonical cover of F is a "minimal" set of functional dependencies equivalent to F, **having no redundant dependencies** or redundant parts of dependencies

Extraneous Attributes

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

Formal Definition for **extraneous Attributes.**

- Consider a set *F* of functional dependencies and the functional dependency $\alpha \rightarrow \beta$ in *F*.
  - Attribute A is **extraneous** in α if $A \in \alpha$ (an attribute A on the left side of $\alpha \rightarrow \beta$)
    and *F* logically implies $(F - \{\alpha \rightarrow \beta\}) \cup \{(\alpha - A) \rightarrow \beta\}$.
  - Attribute *A* is **extraneous** in β if $A \in \beta$ (an attribute A on the right side of $\alpha \rightarrow \beta$) and the set of functional dependencies $(F - \{\alpha \rightarrow \beta\}) \cup \{\alpha \rightarrow (\beta - A)\}$ logically implies *F*.

Extraneous Attributes

- Example: Given $F = \{A \rightarrow C, AB \rightarrow C\}$
  - B is extraneous in $AB \rightarrow C$
  - because $\{A \rightarrow C, AB \rightarrow C\}$ logically implies $A \rightarrow C$
  - (i.e. the result of dropping B from $AB \rightarrow C$).
  - Means B is *left Extraneous* in $AB \rightarrow C$, therefore $AB \rightarrow C$ can be replaced *by* $A \rightarrow C$

- Example: Given $F = \{A \rightarrow C, AB \rightarrow CD\}$
  - C is extraneous in $AB \rightarrow CD$ since $AB \rightarrow C$ can be inferred even after deleting C.
  - Means C is *right extraneous in* $AB \rightarrow CD$, therefore $AB \rightarrow CD$ can be replaced by $AB \rightarrow D$

Testing if an Attribute is Extraneous

- Consider a set *F* of functional dependencies and the functional dependency $\alpha \rightarrow \beta$ in *F*.

- **To test if attribute $A \in \alpha$ is extraneous in α (Left extraneous)**
  1. compute $(\{\alpha\} - A)^+$ using the dependencies in *F*
  2. check that $(\{\alpha\} - A)^+$ **contains** β; if it does, *A* is extraneous in α

- **To test if attribute $A \in \beta$ is extraneous in β (Right extraneous)**
  1. compute α^+ using only the dependencies in F'

$$F' = (F - \{\alpha \rightarrow \beta\}) \cup \{\alpha \rightarrow (\beta - A)\},$$

  2. check that α^+ **contains *A***; if it does, *A* is extraneous in β

Canonical Cover

- A **canonical cover** for *F* is a set of dependencies F_c such that
  - ***F*** logically implies all dependencies in $\boldsymbol{F_c}$, and
  - $\boldsymbol{F_c}$ logically implies all dependencies in ***F***, and
  - No functional dependency in $\boldsymbol{F_c}$ contains an extraneous attribute, and
  - Each left side of functional dependency in $\boldsymbol{F_c}$ is unique.

Canonical Cover: Algorithm

- To compute a canonical cover for *F*:

- $F_c = F$

  repeat

  Use the union rule to replace any dependencies in F_c

  $\alpha_1 \rightarrow \beta_1$ 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: test for extraneous attributes done **using F_c**, not F */

  If an extraneous attribute is found, delete it from $\alpha \rightarrow \beta$

  until *F* does not change

- Note: Union rule may become applicable after some extraneous attributes have been deleted, so it has to be re-applied

Exercises

- Consider a Schema R = (A, B, C, D, E)
- FDs F = {A →B, B →C, C →A, AB →E, BD →A}
- IS AC →B holds ?
- IS C → E holds ?
- IS AE → D holds ?

Exercises

- Consider a Schema R = (A, B, C, D, E)
- FDs F = { A →BCD, BC →DE, B →D, D →A }
- Remove extroneos attributes from both sides.
- A → BCD ,
- BC →DE

To test if attribute $A \in \alpha$ is extraneous in α (Left extraneous)

1. compute $(\{\alpha\} - A)^+$ using the dependencies in *F*
2. check that $(\{\alpha\} - A)^+$ **contains** β; if it does, *A* is extraneous in α

To test if attribute $A \in \beta$ is extraneous in β (Right extraneous)

1. compute α^+ using only the dependencies in F'

$$F' = (F - \{\alpha \rightarrow \beta\}) \cup \{\alpha \rightarrow (\beta - A)\},$$

2. check that α^+ **contains *A***; if it does, *A* is extraneous in β

Exercises

- Consider a Schema R = (A, B, C)
- FDs F = { A →BC, B →C, A→B, AB →C }
- Find Cannonical cover.

To test if attribute $A \in \alpha$ is extraneous in α (Left extraneous)

1. compute $(\{\alpha\} - A)^+$ using the dependencies in *F*
2. check that **$(\{\alpha\} - A)^+$ contains β**; if it does, *A* is extraneous in α

To test if attribute $A \in \beta$ is extraneous in β (Right extraneous)

1. compute α^+ using only the dependencies in F′

   $F' = (F - \{\alpha \rightarrow \beta\}) \cup \{\alpha \rightarrow (\beta - A)\}$,
2. check that **α^+ contains *A*;** if it does, *A* is extraneous in β

$F_c = F$

repeat

Use the union rule to replace any dependencies in F_c

$\alpha_1 \rightarrow \beta_1$ 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: test for extraneous attributes done **using F_c**, not F */

If an extraneous attribute is found, delete it from $\alpha \rightarrow \beta$

until *F* does not change

Computing a Canonical Cover

- $R = (A, B, C)$
  $F = \{A \rightarrow BC,$
  $B \rightarrow C,$
  $A \rightarrow B,$
  $AB \rightarrow C\}$
- Combine $A \rightarrow BC$ and $A \rightarrow B$ into $A \rightarrow BC$
  - Set is now $\{A \rightarrow BC, B \rightarrow C, AB \rightarrow C\}$ **Applying UNION Rule.**
- ***A* is extraneous** in $AB \rightarrow C$ **Checking Left extraneous**
  - Check if the result of deleting A from $AB \rightarrow C$ is implied by the other dependencies
    - Yes: in fact, $B \rightarrow C$ is already present!
  - Set is now $\boldsymbol{\{A \rightarrow BC, B \rightarrow C\}}$
- ***C* is extraneous** in $A \rightarrow BC$ **Checking Right extraneous**
  - Check if $A \rightarrow C$ is logically implied by $A \rightarrow B$ and the other dependencies
    - Yes*:* using transitivity on $A \rightarrow B$ *and* $B \rightarrow$ C.
      - Can use attribute closure of *A* in more complex cases
- The canonical cover is:

$$\mathbf{F_c = \{ A \rightarrow B ,\ B \rightarrow C \}}$$

Exercises

- Consider a Schema R = (A, B, C, D, E)
- FDs F = { A →CD, BC →DE, B →D, D →A }
- Find Cannonical cover.

To test if attribute $A \in \alpha$ is extraneous in α (Left extraneous)

1. compute $(\{\alpha\} - A)^+$ using the dependencies in *F*
2. check that **$(\{\alpha\} - A)^+$ contains β**; if it does, *A* is extraneous in α

To test if attribute $A \in \beta$ is extraneous in β (Right extraneous)

1. compute α^+ using only the dependencies in F′

   $F' = (F - \{\alpha \rightarrow \beta\}) \cup \{\alpha \rightarrow (\beta - A)\}$,
2. check that **α^+ contains *A*;** if it does, *A* is extraneous in β

$F_c = F$

repeat

Use the union rule to replace any dependencies in F_c

$\alpha_1 \rightarrow \beta_1$ 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: test for extraneous attributes done **using F_c**, not F */

If an extraneous attribute is found, delete it from $\alpha \rightarrow \beta$

until *F* does not change

Lossless-join Decomposition

- For the case of **R = (R_1, R_2)**, we require that for all possible relations *r* on schema *R*

$$r = \Pi_{R1}(r) \bowtie \Pi_{R2}(r)$$

- A decomposition of *R* into R_1 and R_2 is lossless join if at least one of the following dependencies is in F^+:
  - $R_1 \cap R_2 \rightarrow R_1$
  - $R_1 \cap R_2 \rightarrow R_2$

This implies common attribute of R1 and R2 must be super key of R_1 or R_2

- The above functional dependencies are a sufficient condition for lossless join decomposition; the dependencies are a necessary condition only if all constraints are functional dependencies

Example

- $R = (A, B, C)$
  $F = \{A \rightarrow B, B \rightarrow C)$
  - Can be decomposed in two different ways
- $\boldsymbol{R_1 = (A, B), \quad R_2 = (B, C)}$
  - Lossless-join decomposition:

    $$\boldsymbol{R_1 \cap R_2 = \{B\} \text{ and } B \rightarrow BC \quad i.e. \; B \rightarrow R_2}$$

  - Dependency preserving
- $\boldsymbol{R_1 = (A, B), \quad R_2 = (A, C)}$
  - Lossless-join decomposition:

    $$\boldsymbol{R_1 \cap R_2 = \{A\} \text{ and } A \rightarrow AB \; i.e. \; A \rightarrow R_1}$$

  - Not dependency preserving
    (cannot check $B \rightarrow C$ without computing $R_1 \bowtie R_2$)

Decomposition: EXAMPLES

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

F = {AB→C, C→E, B→D, E→A}

Assume R is Decomposed into R_1 and R_2 as below-

- R_1(B,C,D) R_2(A,C,E)

Is this a Lossless Decomposition ?Check it.

- R_1(B,E,D) R_2(A,C,E)

Is this a Lossless Decomposition ?Check it.

- R1(A,B,E,C) R2(D,E)

Is this a Lossless Decomposition ?Check it.

- R1(B,C,D) R2(A,D,E)

Is this a Lossless Decomposition ?Check it.

Dependency Preservation

- Let F_i be the set of dependencies F^+ that include only attributes in R_i.
  - A decomposition is **dependency preserving**, if

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

  - If it is not, then checking updates for violation of functional dependencies may require computing joins, which is expensive.

Testing for Dependency Preservation

- To check if a dependency $\alpha \rightarrow \beta$ is preserved in a decomposition of ***R*** **into** $R_1, R_2, \ldots, R_n$ we apply the following test (with attribute closure done with respect to *F*)

```
result = α
while (changes to result) do
        for each Ri in the decomposition
                t = (result ∩ Ri)+ ∩ Ri
                result = result ∪ t
```

  - If ***result*** **contains all attributes in** β, then the functional dependency $\alpha \rightarrow \beta$ **is preserved**.

- **We apply the test on all dependencies in *F* to check if a decomposition is dependency preserving**

- This procedure **takes polynomial time**, instead of the exponential time required to compute F^+ **and** $(F_1 \cup F_2 \cup \ldots \cup F_n)^+$

Dependency Preservation-Example

- Given the following:

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

  F = {AB→C, C→E, B→D, E→A}

  Assume R is Decomposed into R_1 and R_2 as below-

  R_1(B,C,D) R_2(A,C,E)

- **Is this decomposition dependency preserving?**

Example continued…

R(A,B,C,D,E) F = {AB→C, C→E, B→D, E→A}

Decomposition: R1(B,C,D) R2(A,C,E)

Is AB→C Preserved?....

Result=AB

For Result ∩ R1 = AB ∩ BCD = B

$\{B\}^+ = BD$

$\{B\}^+ \cap R1 = BD \cap BCD = BD$

Update Result = AB ∪ BD = ABD,

continue to next for loop

result = α

while (changes to *result*) do

for each R_i in the decomposition

$t = (result \cap R_i)^+ \cap R_i$

$result = result \cup t$

Example continued…

R(A,B,C,D,E) F = {AB→C, C→E, B→D, E→A}

Decomposition: R1(B,C,D) R2(A,C,E)

Is AB→C Preserved? ……

Result=ABD

For Result ∩ R2 = ABD ∩ ACE = A

$\{A\}^+ = A$

$\{A\}^+ \cap R2 = A \cap ACE = A$

Update Result, Result is still ABD

Since Result changed,

continue to next while loop

result = α

while (changes to *result*) do

for each R_i in the decomposition

$t = (result \cap R_i)^+ \cap R_i$

$result = result \cup t$

Example continued…

R(A,B,C,D,E) F = {AB→C, C→E, B→D, E→A}

Decomposition: R1(B,C,D) R2(A,C,E)

Is AB→C Preserved? ……

Result=ABD

For Result ∩ R1 = ABD ∩ BCD = BD

$\{BD\}^+ = BD$

$\{BD\}^+ \cap R1 = BD \cap BCD = BD$

Update Result = ABD ∪ BD = ABD,

Continue to next for loop

result = α

while (changes to *result*) do

for each R_i in the decomposition

$t = (result \cap R_i)^+ \cap R_i$

$result = result \cup t$

Example

R(A,B,C,D,E) F = { AB→C, C→E, B→D, E→A }

Decomposition: R1(B,C,D) R2(A,C,E)

List the functional dependencies which are not preserved?

Is AB→C Preserved?

Result=ABD

For Result ∩ R2 = ABD ∩ ACE = A

$\{A\}^+ = A$

$\{A\}^+ \cap R2 = A \cap ACE = A$

Update Result, Result is still ABD

Since Result hasn't changed, we stop here.

Check, Is Result Contains C ? **NO**

Therefore we can conclude

AB→C is not preserved.

Similarly check other functional dependencies.

Example continued…

R(A,B,C,D,E) F = {AB→C, C→E, B→D, E→A}

Decomposition: R1(B,C,D) R2(A,C,E)

Is B→D Preserved ?.....

Result=B

For Result ∩ R1 = B ∩ BCD = B

$\{B\}^+ = BD$

$\{B\}^+ \cap R1 = BD \cap BCD = BD$

Update Result = B ∪ BD = BD

Continue to next for loop

result = α

while (changes to *result*) do

for each R_i in the decomposition

$t = (result \cap R_i)^+ \cap R_i$

$result = result \cup t$

Example continued…

R(A,B,C,D,E) F = {AB→C, C→E, B→D, E→A}

Decomposition: R1(B,C,D) R2(A,C,E)

Is B→D Preserved ?

Result=BD

For Result ∩ R2 = BD ∩ ACE = Φ

$\{\Phi\}^+ = \Phi$

$\{\Phi\}^+ \cap R2 = \Phi \cap ACE = \Phi$

Update Result = B ∪ Φ = BD

Continue to next while loop.

Result unchanged. Stop

Since D belongs to Result,

B→D is preserved.

result = α

while (changes to *result*) do

for each R_i in the decomposition

$t = (result \cap R_i)^+ \cap R_i$

$result = result \cup t$

Example continued…

R(A,B,C,D,E) F = {AB→C, C→E, B→D, E→A}

Decomposition: R1(B,C,D) R2(A,C,E)

Is C→E Preserved ?....

Result=C

For Result ∩ R1 = C ∩ BCD = C

$\{C\}^+ = CEA$

$\{C\}^+ \cap R1 = CEA \cap BCD = C$

Update Result = C ∪C= C

Continue to next for loop

result = α

while (changes to *result*) do

for each R_i in the decomposition

$t = (result \cap R_i)^+ \cap R_i$

$result = result \cup t$

Example continued…

R(A,B,C,D,E) F = {AB→C, C→E, B→D, E→A}

Decomposition: R1(B,C,D) R2(A,C,E)

Is C→E Preserved ?

Result=C

For Result ∩ R2 = C ∩ ACE = C

$\{C\}^+ = CEA$

$\{C\}^+ \cap R2 = CEA \cap ACE = ACE$

Update Result = C ∪ACE= ACE

In next while loop result remains unchanged hence stop.

Since E is in Result i.e. ACE,

C→E is preserved.

Example continued…

R(A,B,C,D,E) F = {AB→C, C→E, B→D, E→A}

Decomposition: R1(B,C,D) R2(A,C,E)

Is E→A Preserved ?.....

Result=E

For Result $\cap$ R1 = E $\cap$ BCD = Φ $\{\Phi\}^+ = \Phi$

$\{\Phi\}^+ \cap$ R1 = $\Phi \cap$ BCD = Φ

Update Result = E $\cup$ Φ = E

Continue to next for loop

Example continued…

R(A,B,C,D,E) F = {AB→C, C→E, B→D, E→A}

Decomposition: R1(B,C,D) R2(A,C,E)

Is E→A Preserved ?

Result=E

For Result ∩ R2 = E ∩ ACE = E

$\{E\}^+ = EA$

$\{E\}^+ \cap R1 = EA \cap ACE = EA$

Update Result = E ∪ EA= EA

In next while loop result remains unchanged hence stop.

Since A belongs to Result i.e. EA,

E→A is preserved.

Example continued…

R(A,B,C,D,E) F = {AB→C, C→E, B→D, E→A}

Decomposition: R1(B,C,D) R2(A,C,E)

AB→C is not preserved.

C→E is preserved.

B→D is preserved

E→A is preserved.

As one of the functional dependencies AB→C is not is not preserved, Therefore decomposition R1,R2 is not a dependency preserving decomposition.

First Normal Form

- A relational schema R is in **first normal form** if the domains of all attributes of R are atomic
- Domain is **atomic** if its elements are considered to be indivisible units
  - Examples of **non-atomic** domains:
    - Set of names, composite attributes
    - Identification numbers like "CS101" that can be broken up into parts-Dept & RollNo
- **Non-atomic** values complicate storage and encourage redundant (repeated) storage of data
  - Example: Set of accounts stored with each customer, and set of owners stored with each account

Second Normal Form (2NF)

For a table to be in 2NF, there are two requirements

- The database is in first normal form
- All **nonkey** attributes in the table must be fully functionally dependent on the entire primary key

Example (Not 2NF)

Scheme , Book_author(PubId, AuId, Title, Price, AuAddress)

1. Primary Key → {PubId, AuId}
2. {PubId, AuID} → {Price}
3. {PubId, AuID} → {Title}
4. {AuID} → {AuAddress}
5. AuAddress is not determined by all key attributes
6. AuAddress functionally depends on AuId which is a subset of a key

2NF - Decomposition

1. If a data item is fully functionally dependent on only a part of the primary key, move that data item and that part of the primary key to a new table.
2. If other data items are functionally dependent on the same part of the key, place them in the new table also
3. Make the partial primary key copied from the original table the primary key for the new table. Place all items that appear in the repeating group in a new table

Example 1 (Convert to 2NF)

Old Scheme {PubId, AuId, Title ,Price, AuAddress}

New Scheme1 {PubId, AuId, Title, Price}

New Scheme 2 {AuId, AuAddress}

3 NF

A relation schema R is in **third normal form** 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**.
- α is a **super key** for R.
- Each attribute A **in** $\beta - \alpha$ is contained in a candidate key for R.

Note that the third condition above does not say that a single candidate key must contain all the attributes in $\beta - \alpha$; **each attribute A** in $\beta - \alpha$ may be **contained in a different candidate key.**

Any **"schema that satisfies BCNF also satisfies 3NF"**, since each of its functional dependencies would satisfy one of the first two alternatives. **BCNF is therefore a more restrictive** normal form than is 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 super key.**
- If α is **not a super key**, we have to **verify if each attribute in $\beta - \alpha$ is contained in a candidate key of *R***
  - Attributes in $\beta - \alpha$ may be in different candidate keys of R

3NF Example

- Relation ***dept_advisor***: (refer slide for dept_advisor example)
  - ***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 dept_name)* is a super key
    - *i_ID* $\rightarrow$ *dept_name* ; *i_ID is not super key* , *but*
      - *dept_name* is contained in a candidate key (s_ID dept_name)

Refer slide for 3NF

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**

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 */

$R_j = R_i$;

$i=i-1$;

return $(R_1, R_2, ..., R_i)$

Note: algorithm ensures:

- each relation schema R_i is in 3NF
- decomposition is dependency preserving and lossless-join

3NF Decomposition example

Assume $R = (A, B, C)$

$F_c = \{A \rightarrow B\ ,\ B \rightarrow C\}$, A is candidate key (if not given find yourself).

R is not in 3 NF and Decompose to 3 NF

Solution:

i=0

For loop

take $A \rightarrow B$, i=1 , R_1=A,B

take $B \rightarrow C$, i=2 , R_2 =B,C

IF none of R_1 ,R_2 contains Candidate key (i.e. A) is FALSE

because A is contained in R_1

Repeat

there is neither R_1 contained in R_2 nor R_2 contained in R_1

therefore No Deletion of duplicate schema

Return (R_1,R_2)

"R_1(A,B) & R_2(B,C) are decomposed 3 NF Relations"

3NF Decomposition: An Example

- **Relation schema:**

  cust_banker_branch = (customer_id, employee_id, branch_name, type)

  Assume (customer_id, employee_id) is the candidate key*

- The **functional dependencies** for this relation schema are:

  1. customer_id, employee_id → branch_name, type
  2. employee_id → branch_name
  3. customer_id, branch_name → 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 → type
  employee_id → branch_name
  customer_id, branch_name → employee_id }

3NF Decompsition Example (Cont.)

- 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 namely-(customer_id, branch_name, employee_id)
  - result will not depend on the order in which FDs are considered

- The resultant simplified 3NF schema is:

  R_1 (customer_id, employee_id, type)

  R_2 (customer_id, branch_name, employee_id)

R(A,B,C,D,E,F,G,H) and F={ ABC →DE , E →BCG, F →AH }

Is it in 3NF? Decompose into 3NF.

- Assume candidate keys are FE and FBC
- F_c ={ABC →DE , E →BCG, F →AH }
-

R(A,B,C,D,E,F,G,H,I,J) and

F={ AB → C ,A → DE , B →F, F →GH, D →IJ}

Is it in 3NF? Decompose into 3NF.

- Assume candidate keys are AB
- F_c={ AB → C ,A → DE , B →F, F →GH, D →IJ}

Test for 3NF-example

$R = (J, K, L)$

$F = \{JK \rightarrow L, L \rightarrow K\}$ **is R in 3NF ?**

- Take $JK \rightarrow L$, *is JK* is super key ?

  $JK^+ = \{ JKL\}$, yes, JK is super key, **1st condition for 3NF satisfied**

  Note JK is candidate key also (check yourself)

- Take $L \rightarrow K$ *is L* is super key ?

  $L^+ = \{ L\}$, no, L is not super key, **1st condition for 3NF NOT satisfied**

  K-L =K & K contained in candidate key (JK), **2nd condition for 3NF satisfied.**

"Therefore *R (J, K, L) is in 3NF*"

If you Replace J=s_ID ; K=dept_name & L= i_ID , this example is same as given in slide

Redundancy in 3NF

- There is some redundancy in this schema

  (the relation dept_advisor given in slide can also be discussed as below)

- **Example for problems due to redundancy in 3NF**
  - $R = (J, K, L)$
    $F = \{JK \rightarrow L, L \rightarrow K\}$

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

If you Replace J=s_ID ; K=dept_name & L= i_ID , this example is same as given in slide 75

- repetition of information (e.g., the relationship l_1, k_1)
  - i.e. (*i_ID, dept_name)* *in slide 75*
- need to use null values (e.g., to represent the relationship l_2, k_2 where there is no corresponding value for J).

(i.e. for an instructor in a department may not be an advisor for any student ,S_Id (i.e. J here is Null)

  - (*i_ID, dept_nameI)* if there is no separate relation mapping instructors to departments

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 trivial (i.e., $\beta \subseteq \alpha$)
- α is a super key for *R*

Example schema *not* in BCNF:

instr_dept (*ID, name, salary, dept_name, building, budget*)

Assume F={ *ID* → *name, salary , dept_name, dept_name* → *building, budget* }

instr_dept is not in BCNF because *dept_name* → (*building, budget)*

BECAUSE, ***dept_name* is not a superkey**

Decomposing a Schema into BCNF

- Suppose we have a schema *R* and a non-trivial dependency $\alpha \rightarrow \beta$ causes a violation of BCNF.

  We decompose *R* into:

  - $(\alpha \cup \beta)$
  - $(R - (\beta - \alpha))$

On decomposing a Relation R ,one or more of the resulting schemas may not be in BCNF. In such cases, further decomposition is required, the eventual result of which is a set of BCNF schemas.

- In our example,
- **Inst_dept (ID, Name, Salary, Dept_Name, Building, Budget)**
- **Assume** ***dept_name →building, budget*** *violates*
  - α = *dept_name* β = *building, budget*

    and ***inst_dept*** is replaced by
  - $(\alpha \cup \beta)$ = (*dept_name, building, budget*)
  - (*R* - (β - α)) = (*ID, name, salary, dept_name*)
  - **R_1(*dept_name, building, budget*)**
  - **R_2(*ID, name, salary, dept_name*)**

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 α is a super key of *R*.
  - It suffices to **check only the dependencies in the given set F** for violation of BCNF, rather than check all dependencies in $\mathbf{F}^+$.
  - We can show that if none of the dependencies in F causes a violation of BCNF, then none of the dependencies in $\mathbf{F}^+$ will cause a violation of BCNF,either.
- However, procedure does not work **when a relation is decomposed**.
- That is, **it *does not* suffice** to use *F* when we test a relation R_i , in a decomposition of *R*, for violation of BCNF.

Example

- $R = (A, B, C)$
  $F = \{A \rightarrow B,\ B \rightarrow C\}$
  Key = $\{A\}$
- R is not in BCNF (why?)
- Assume R is Decomposed into $R_1 = (A, B)$, $R_2 = (B, C)$
  - R_1 and R_2 in BCNF (check*) (see notes section)s
  - Lossless-join decomposition (check)
  - Dependency preserving (check)

* Using F it is not possible to check whether R1 and R2 are in BCNF. We need functional dependency set F1,F2 corresponding to R1 and R2 respectively.

Testing Decomposition for BCNF

when R is decomposed into R_i

- To check if a relation R_i in a decomposition of R is in BCNF,
  - Either test 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: (**this method we use in further discussion**)
    - 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.

Example: $R = (A, B, C)$

$F = \{A \rightarrow B,\ B \rightarrow C\}$,

R is not in BCNF and we got Decomposition $R_1 = (A, B)$, $R_2 = (B, C)$

Is R_1 and R_2 in BCNF ?

Example: $R = (A, B, C)$

$F = \{A \rightarrow B,\ B \rightarrow C\}$,

R is not in BCNF and we got Decomposition

$R_1 = (A, B)$, $R_2 = (B, C)$ **Is R_1 and R_2 in BCNF ?**

Take R_1 all subsets of R_1 are A , B ,AB

1. $A^+ = \{A,B,C\}$ $R_1 - A = B$

A^+ do not include any attribute of **$R_1 - A$ is- FALSE**, but

A^+ include ALL attributes of R_1 is- TRUE

2. $B^+ = \{B,C\}$ $R_1 - B = A$

B^+ do not include any attribute of **$R_1 - B$ is- TRUE**

R_1 is in BCNF

α^+ either includes **no** attribute of $R_i - \alpha$,
or includes **all** attributes of R_i.

Example: continued..

$R = (A, B, C)$

$F = \{A \rightarrow B,\ B \rightarrow C\}$,

R is not in BCNF and we got Decomposition

$R_1 = (A, B)$, $R_2 = (B, C)$ **Is R_1 and R_2 in BCNF ?**

Take R_2 all subsets of R_2 are B,C ,BC

1. $B^+ = \{B,C\}$ $R_2 - B = C$

B^+ do not include any attribute of **$R_2 - B$ is- FALSE**,

but

B^+ include ALL attributes of R_2 is- TRUE

2. $C^+ = \{C\}$

C^+ do not include any attribute of **$R_2 - C$ is- TRUE**

R_2 is in BCNF

α^+ either includes no attribute of $R_i - \alpha$,
or includes all attributes of R_i.

BCNF Decomposition Algorithm

A **general method to decompose** a relation schema so as to satisfy BCNF.

result := { *R* } ;
done := false ;
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 α^+ **does not in** R_i **AND** $\alpha \cap \beta = \varnothing$
          ***result* := (*result* – R_i) $\cup$ (R_i – β) $\cup$ (α, β) ;**

      end
    else *done* := **true ;**

.

The decomposition that this algorithm generates is **not only in BCNF**, but is also a **lossless decomposition**.

Example of BCNF Decomposition

Suppose we have a schema R and a non-trivial dependency $\alpha \rightarrow \beta$ causes a violation of BCNF.

According to algorithm, we decompose R into:

$$result := (result - R_i) \cup (R_i - \beta) \cup (\alpha , \beta)$$

- ***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 (Cont.)

- BCNF Decomposition:
  - ***course_id*** $\rightarrow$ ***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 (Cont.)

- *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.

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

BCNF and Dependency Preservation

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

- $R = (J, K, L)$

  $F = \{ JK \rightarrow L , \ L \rightarrow K \}$

  Two candidate keys = *JK* and *JL* *(see the similar example in slide)*

- ***R*** is **not in BCNF**
- As $L \rightarrow K$ ***fails*** , *we have to decompose into* $R_1(J,L)$ & $R_2(L,K)$
- Any decomposition of *R* will fail to preserve

$$JK \rightarrow L$$

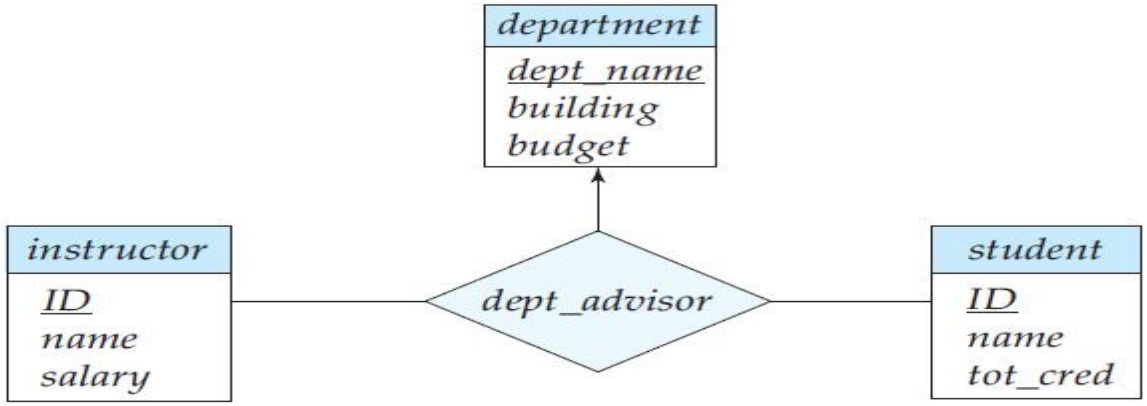
This implies that testing for $JK \rightarrow L$ requires a join

This example illustrate that when R is decomposed in to R_1 & R_2 satisfying BCNF, always it may not be Dependency Preserving.

"BCNF decomposition is not Always Dependency Preserving"

This is the Motivation for 3NF

This example also illustrates BCNF is Not always Dependency Preserving



The schema derived from *the ER diagram* is

Instructor(ID, name, salary) ***Student(ID, name, tot_cred)***

Dept_advisor (s ID, i ID, deptname)

As per the constraints the following functional dependencies hold on ***dept_advisor***:

i_ID→dept name

S_ID, deptname→i ID

Dept_advisor

S_Id	i_Id	DeptName
S1	I1	D1
S1	I2	D2
S2	I1	D1
S2	I2	D2
S3	I3	D1
NULL	I4	D1

Note: For a Instructor **I4** belonging to DeptName **D1** mqy not be advisor for any of the student ,so **S_id is NULL** for it.

- Note this is not BCNF
- As per decomposition rule we get following schema(for dept_advisor)-
  - **(*s_ID*, *i ID*)**
  - **(*i_ID*, *dept name*)**
- In this schema is BCNF but,
- we are unable to represent ***(S_ID, dept name) →i_ID FD.***
- ***Therefore FD is not preserved in BCNF.***

BCNF and Dependency Preservation

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

- $R = (J, K, L)$

  $F = \{ JK \rightarrow L , \ L \rightarrow K \}$

  Two candidate keys = *JK* and *JL* *(see the similar example in slide)*

- ***R*** is **not in BCNF**
- As $L \rightarrow K$ ***fails*** , *we have to decompose into* $R_1(J,L)$ & $R_2(L,K)$
- Any decomposition of *R* will fail to preserve

$$JK \rightarrow L$$

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

This example illustrate that when R is decomposed in to R_1 & R_2 satisfying BCNF, always it may not be Dependency Preserving.

"BCNF decomposition is not Always Dependency Preserving"

This is the Motivation for 3NF

END

END

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 trivial (i.e., $\beta \subseteq \alpha$)
- α is a super key for *R*

Example schema *not* in BCNF:

instr_dept (*ID, name, salary, dept_name, building, budget*)

Assume F={ *ID* → *name, salary , dept_name, dept_name* → *building, budget* }

instr_dept is not in BCNF because *dept_name*→ (*building, budget*) holds on ***instr_dept***, but ***dept_name* is not a superkey**

Decomposing a Schema into BCNF

- Suppose we have a schema *R* and a non-trivial dependency $\alpha \rightarrow \beta$ causes a violation of BCNF.

  We decompose *R* into:

  - $(\alpha \cup \beta)$
  - $(R - (\beta - \alpha))$

On decomposing a Relation R ,one or more of the resulting schemas may not be in BCNF. In such cases, further decomposition is required, the eventual result of which is a set of BCNF schemas.

- In our example,
- **Inst_dept (ID, Name, Salary, Dept_Name, Building, Budget)**
- **Assume** ***dept_name →building, budget*** *violates*
  - α = *dept_name* β = *building, budget*

    and ***inst_dept*** is replaced by
  - $(\alpha \cup \beta)$ = (*dept_name, building, budget*)
  - (*R* - (β - α)) = (*ID, name, salary, dept_name*)
  - **R_1(*dept_name, building, budget*)**
  - **R_2(*ID, name, salary, dept_name*)**

BCNF and Dependency Preservation

- Constraints*, including functional dependencies, are costly to check in practice unless they pertain to only one relation
- If it is sufficient to test only those dependencies on each individual relation of a decomposition in order to ensure that *all* functional dependencies hold, then that decomposition is *dependency preserving.*
- Because it is not always possible to achieve both BCNF and dependency preservation, we consider a weaker normal form, known as *third normal form.*
  - suppose that we make a small change to our university Organization, see the ER diagram in next slide.

    The "small" change we shall make is that an instructor can be associated with only a single department and a student may have more than one advisor, but at most one from a given department.

** See the notes section*

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,**

     α is a super key of *R*.
  - It suffices to check only the dependencies in the given set F for violation of BCNF, rather than check all dependencies in $\mathbf{F}^+$.
  - We can show that if none of the dependencies in F causes a violation of BCNF, then none of the dependencies in $\mathbf{F}^+$ will cause a violation of BCNF,either.
- However, procedure does not work **when a relation is decomposed**.
- That is, **it *does not* suffice** to use *F* when we test a relation R_i , in a decomposition of *R*, for violation of BCNF.

Example

- $R = (A, B, C)$
  $F = \{A \rightarrow B,\ B \rightarrow C\}$
  Key = $\{A\}$
- R is not in BCNF (why?)
- Assume R is Decomposed into $R_1 = (A, B)$, $R_2 = (B, C)$
  - R_1 and R_2 in BCNF (check*) (see notes section)s
  - Lossless-join decomposition (check) (slide 47 example)
  - Dependency preserving (check)

* Using F it is not possible to check whether R1 and R2 are in BCNF. We need functional dependency set F1,F2 corresponding to R1 and R2 respectively.

Testing Decomposition for BCNF

when R is decomposed into R_i

- To check if a relation R_i in a decomposition of R is in BCNF,
  - Either test 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: (**this method we use in further discussion**)
    - 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.

Example: $R = (A, B, C)$

$F = \{A \rightarrow B,\ B \rightarrow C\}$,

R is not in BCNF and we got Decomposition $R_1 = (A, B)$, $R_2 = (B, C)$

Is R_1 and R_2 in BCNF ?

Example: $R = (A, B, C)$

$F = \{A \rightarrow B,\ B \rightarrow C\}$,

R is not in BCNF and we got Decomposition

$R_1 = (A, B)$, $R_2 = (B, C)$ Is R_1 and R_2 in BCNF ?

Take R_1 all subsets of R_1 are A , B ,AB

1. $A^+ = \{A,B,C\}$ $R_1 - A = B$

A^+ do not include any attribute of **$R_1 - A$ is FALSE**, but

A^+ include ALL attributes of R_1 is TRUE

2. $B^+ = \{B,C\}$ $R_1 - B = A$

B^+ do not include any attribute of **$R_1 - B$ is TRUE**

R_1 is in BCNF

Take R_2 all subsets of R_2 are B,C ,BC

1. $B^+ = \{B,C\}$ $R_2 - B = C$

B^+ do not include any attribute of **$R_2 - B$ is FALSE**, but

B^+ include ALL attributes of R_2 is TRUE

2. $C^+ = \{C\}$

C^+ do not include any attribute of **$R_2 - C$ is TRUE**

R_2 is in BCNF

BCNF Decomposition Algorithm

A **general method to decompose** a relation schema so as to satisfy BCNF.

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** $\alpha \cap \beta = \varnothing$
          ***result* := (*result* – R_i) $\cup$ (R_i – β) $\cup$ (α, β) ;**

      end
    else *done* := **true ;**

The decomposition that this algorithm generates is **not only in BCNF**, but is also a **lossless decomposition**.

BCNF Decomposition Algorithm

A **general method to decompose** a relation schema so as to satisfy BCNF.

result := { *R* } ;
done := false ;
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 α^+ **does not in** R_i **AND** $\alpha \cap \beta = \varnothing$
          ***result* := (*result* – R_i) $\cup$ (R_i – β) $\cup$ (α, β) ;**

      end
    else *done* := **true ;**

.

The decomposition that this algorithm generates is **not only in BCNF**, but is also a **lossless decomposition**.

BCNF decomposition Example

Assume $R = (A, B, C)$

$F = \{A \rightarrow B,\ B \rightarrow C\}$,

R is not in BCNF and Decompose to BCNF

Initially as there are result= {R} & Done:=False

Calculate F^+ refer slide 40 for calculating F^+

$F^+ = \{ A \rightarrow A, B \rightarrow B, C \rightarrow C, A \rightarrow B, A \rightarrow C, A \rightarrow AB, A \rightarrow AC, A \rightarrow BC,$
$A \rightarrow ABC, B \rightarrow C, B \rightarrow BC, AB \rightarrow B, AB \rightarrow C, AB \rightarrow AB, AB \rightarrow AC, AB \rightarrow BC,$
$AB \rightarrow ABC, AC \rightarrow B, AC \rightarrow C, AC \rightarrow AB, AC \rightarrow AC, AC \rightarrow BC, AC \rightarrow ABC,$
$ABC \rightarrow B, ABC \rightarrow C, ABC \rightarrow AB, ABC \rightarrow AC, ABC \rightarrow BC, ABC \rightarrow ABC \}$

Non-Trivial FD from F+

$F_1^+ = \{A \rightarrow B, A \rightarrow C, A \rightarrow AB, A \rightarrow AC, A \rightarrow BC, A \rightarrow ABC,$
$B \rightarrow C, B \rightarrow BC, AB \rightarrow C, AB \rightarrow AC, AB \rightarrow BC, AB \rightarrow ABC, AC \rightarrow B, AC \rightarrow$
$AB, AC \rightarrow BC, AC \rightarrow ABC \}$

…BCNF decomposition Example

$\alpha \rightarrow R_i$ is not in F^+

In F_1^+ find of functional dependencies of type $\alpha \rightarrow R_i$ *means where* α is super key for the decomposition R_i , *in fact in this example yet (in the beginning) R is not decomposed ,therefore R is taken.*

Find all functional dependencies in which *LHS (i.e.* α) is super key and remove all such functional dependencies from F_1^+

In this example A is the key and hence any attribute in combination with A will be super keys.

Therefore we can remove $A \rightarrow B$, $A \rightarrow C$, $A \rightarrow AB$, $A \rightarrow AC$, $A \rightarrow BC$, $A \rightarrow ABC$, $AB \rightarrow C$,$AB \rightarrow AC$, $AB \rightarrow BC$, $AB \rightarrow ABC$, $AC \rightarrow B$, $AC \rightarrow AB$, $AC \rightarrow BC$, $AC \rightarrow ABC$ from F_1^+ and let us call as F_2^+ **(see next slide)**

…BCNF decomposition Example

$F_2^+ = \{B \rightarrow C, B \rightarrow BC\}$

In these functional dependencies (F_2^+) take functional dependency $\alpha \rightarrow \beta$ **such that** $\alpha \cap \beta = \varnothing$

From F_2^+

for $B \rightarrow C$ $\quad B \cap C = \varnothing$

for $B \rightarrow BC$ $\quad B \cap BC \neq \varnothing$

Therefore only $B \rightarrow C$ **is used in the algorithm to test**

$$result := (result - R_i) \cup (R_i - \beta) \cup (\alpha, \beta) \ ;$$

In the beginning as there are no decompositions (R_i) , result={R} itself.

$$result := (R - R) \cup (R - C) \cup (B,C)$$

$$result := \varnothing \cup (A,B) \cup (B,C)$$

Now result is two decomposed relations as below-

result:= { (A,B) , (B,C)}

Therefore R_1 (A,B) , R_2(B,C)

In the next while loop , check R_1 for BCNF , it is in BCNF , so retain as it is.

In next while loop ,check R_2 for BCNF , it is also in BCNF , so retain as it is.

Finally in the ELSE part done:= TRUE and exits from algorithm

Decomposition into BCNF (SIMPLE APPROACH)

Given A Relation R with FD's F

- Look among the given FD's for a BCNF violation $X \rightarrow Y$
- Compute X^+

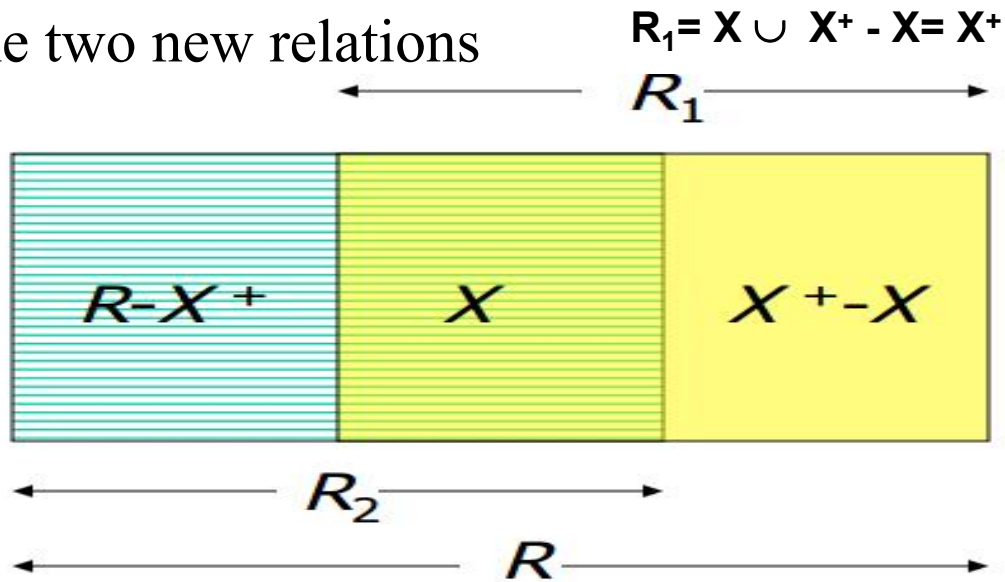
Not all attributes proceed to decomposition , or else X is a super key (so it will be satisfying BCNF)

Decompose R Using $X \rightarrow Y$

Replace R by relations with schemas:

1. $R_1 = X^+$ (note X^+ also includes Y)
2. $R_2 = R - (X^+ - X)$

Project given FD's F onto the two new relations



Example of BCNF Decomposition

Suppose we have a schema R and a non-trivial dependency $\alpha \rightarrow \beta$ causes a violation of BCNF.

According to algorithm, we decompose R into:

$$result := (result - R_i) \cup (R_i - \beta) \cup (\alpha , \beta)$$

- ***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 (Cont.)

- BCNF Decomposition:
  - ***course_id*** $\rightarrow$ ***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 (Cont.)

- *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.

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

END

BCNF and Dependency Preservation

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

- $R = (J, K, L)$

  $F = \{ JK \rightarrow L , \ L \rightarrow K \}$

  Two candidate keys = *JK* and *JL* *(see the similar example in slide)*

- ***R*** is **not in BCNF**
- As $L \rightarrow K$ ***fails*** , *we have to decompose into* $R_1(J,L)$ & $R_2(L,K)$
- Any decomposition of *R* will fail to preserve

$$JK \rightarrow L$$

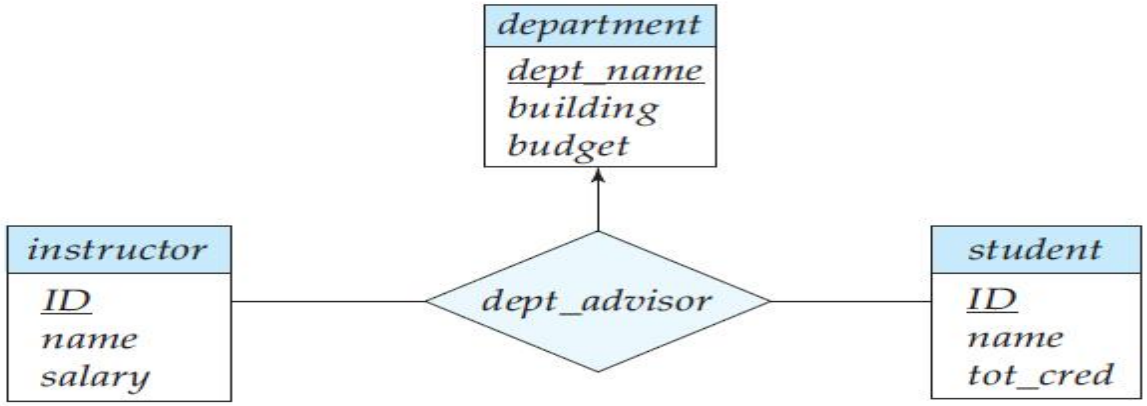
This implies that testing for $JK \rightarrow L$ requires a join

This example illustrate that when R is decomposed in to R_1 & R_2 satisfying BCNF, always it may not be Dependency Preserving.

"BCNF decomposition is not Always Dependency Preserving"

This is the Motivation for 3NF

This example also illustrates BCNF is Not always Dependency Preserving



The schema derived from *the ER diagram* is

Instructor(ID, name, salary) ***Student(ID, name, tot_cred)***

Dept_advisor (s ID, i ID, deptname)

As per the constraints the following functional dependencies hold on ***dept_advisor***:

i_ID→dept name

S_ID, deptname→i ID

Dept_advisor

S_Id	i_Id	DeptName
S1	I1	D1
S1	I2	D2
S2	I1	D1
S2	I2	D2
S3	I3	D1
NULL	I4	D1

Note: For a Instructor **I4** belonging to DeptName **D1** mqy not be advisor for any of the student ,so **S_id is NULL** for it.

- Note this is not BCNF
- As per decomposition rule we get following schema(for dept_advisor)-
  - **(*s_ID*, *i ID*)**
  - **(*i_ID*, *dept name*)**
- In this schema is BCNF but,
- we are unable to represent ***(S_ID, dept name) →i_ID FD.***
- ***Therefore FD is not preserved in BCNF.***

3 NF

A relation schema R is in **third normal form** 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**.
- α is a **super key** for R.
- Each attribute A **in** $\beta - \alpha$ is contained in a candidate key for R.

Note that the third condition above does not say that a single candidate key must contain all the attributes in $\beta - \alpha$; **each attribute A** in $\beta - \alpha$ may be **contained in a different candidate key.**

Any **"schema that satisfies BCNF also satisfies 3NF"**, since each of its functional dependencies would satisfy one of the first two alternatives. BCNF is therefore a more restrictive normal form than is 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 super key.**
- If α is **not a super key**, we have to **verify if each attribute in $\beta - \alpha$ is contained in a candidate key of *R***
  - Attributes in $\beta - \alpha$ may be in different candidate keys of R

3NF Example

- Relation ***dept_advisor***: (refer slide for dept_advisor example)
  - ***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 dept_name)* is a superkey
    - *i_ID* $\rightarrow$ *dept_name* ; *i_ID is not super key* , *but*
      - *dept_name* is contained in a candidate key (s_ID dept_name)

Refer slide for 3NF

Test for 3NF-example

$R = (J, K, L)$

$F = \{JK \rightarrow L, L \rightarrow K\}$ **is R in 3NF ?**

- Take $JK \rightarrow L$, *is JK* is super key ?

  $JK^+ = \{ JKL\}$, yes, JK is super key, **1st condition for 3NF satisfied**

  Note JK is candidate key also (check yourself)

- Take $L \rightarrow K$ *is L* is super key ?

  $L^+ = \{ L\}$, no, L is not super key, **1st condition for 3NF NOT satisfied**

  K-L =K & K contained in candidate key (JK), **2nd condition for 3NF satisfied.**

"Therefore *R (J, K, L) is in 3NF*"

If you Replace J=s_ID ; K=dept_name & L= i_ID , this example is same as given in slide

Redundancy in 3NF

- There is some redundancy in this schema

  (the relation dept_advisor given in slide can also be discussed as below)

- **Example for problems due to redundancy in 3NF**
  - $R = (J, K, L)$
    $F = \{JK \rightarrow L, L \rightarrow K\}$

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

If you Replace J=s_ID ; K=dept_name & L= i_ID , this example is same as given in slide 75

- repetition of information (e.g., the relationship l_1, k_1)
  - i.e. (*i_ID, dept_name)* *in slide 75*
- need to use null values (e.g., to represent the relationship l_2, k_2 where there is no corresponding value for J).

(i.e. for an instructor in a department may not be an advisor for any student ,S_Id (i.e. J here is Null)

  - (*i_ID, dept_nameI)* if there is no separate relation mapping instructors to departments

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**

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 */

$R_j = R_i$;

$i=i-1$;

return $(R_1, R_2, ..., R_i)$

Note: algorithm ensures:

- each relation schema R_i is in 3NF
- decomposition is dependency preserving and lossless-join

3NF Decomposition example

Assume $R = (A, B, C)$

$F_c = \{A \rightarrow B\ ,\ B \rightarrow C\}$, **A is candidate key (if not given find yourself).**

***R* is not in 3 NF and Decompose to 3 NF**

Solution:

i=0

For loop

take $A \rightarrow B$, i=1 , R_1=A,B

take $B \rightarrow C$, i=2 , R_2 =B,C

IF none of R_1 ,R_2 contains Candidate key (i.e. A) is FALSE

because A is contained in R_1

Repeat

there is neither R_1 contained in R_2 nor R_2 contained in R_1

therefore No Deletion of duplicate schema

Return (R_1,R_2)

"R_1(A,B) & R_2(B,C) are decomposed 3 NF Relations"

3NF Decomposition: An Example

- Relation schema:

  cust_banker_branch = (*customer_id, employee_id, branch_name, type*)

  *Assume (customer_id, employee_id) is the candidate key**

- The functional dependencies for this relation schema are:
  1. *customer_id, employee_id* → *branch_name, type*
  2. *employee_id* → *branch_name*
  3. *customer_id, branch_name* → *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* → *type*
employee_id → *branch_name*
customer_id, branch_name → *employee_id* }

* *Finding candidate key* refer slide

3NF Decompsition Example (Cont.)

- 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 namely- (*customer_id, branch_name, employee_id)*

  - 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)*

R(A,B,C,D,E,F,G,H) and F={ ABC →DE , E →BCG, F →AH }

Is it in 3NF? Decompose into 3NF.

- Assume candidate keys are FE and FBC
- F_c ={ABC →DE , E →BCG, F →AH }
-

R(A,B,C,D,E,F,G,H,I,J) and

F={ AB → C ,A → DE , B →F, F →GH, D →IJ}

Is it in 3NF? Decompose into 3NF.

- Assume candidate keys are AB
- F_c={ AB → C ,A → DE , B →F, F →GH, D →IJ}

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.

Design Goals

- Goal for a relational database design is:
  - BCNF.
  - Lossless join.
  - Dependency preservation.
- If we cannot achieve this, we accept one of
  - Lack of dependency preservation
  - Redundancy due to use of 3NF
- Interestingly, SQL does not provide a direct way of specifying functional dependencies other than superkeys.

  Can specify FDs using assertions, but they are expensive to test, (and currently not supported by any of the widely used databases!)

3 NF

A relation schema R is in **third normal form** 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**.
- α is a **super key** for R.
- Each attribute A **in** $\beta - \alpha$ is contained in a candidate key for R.

Note that the third condition above does not say that a single candidate key must contain all the attributes in $\beta - \alpha$; each attribute A in $\beta - \alpha$ may be **contained in a different candidate key.**

Any **"schema that satisfies BCNF also satisfies 3NF"**, since each of its functional dependencies would satisfy one of the first two alternatives. BCNF is therefore a more restrictive normal form than is 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 super key.**
- If α is **not a super key**, we have to **verify if each attribute in $\beta - \alpha$ is contained in a candidate key of *R***
  - Attributes in $\beta - \alpha$ may be in different candidate keys of R

3NF Example

- Relation ***dept_advisor***: (refer slide for dept_advisor example)
  - ***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 dept_name)* is a superkey
    - *i_ID* $\rightarrow$ *dept_name* ; *i_ID is not super key* , *but*
      - *dept_name* is contained in a candidate key (s_ID dept_name)

Refer slide for 3NF

Test for 3NF-example

$R = (J, K, L)$

$F = \{JK \rightarrow L, L \rightarrow K\}$ **is R in 3NF ?**

- Take $JK \rightarrow L$, *is JK* is super key ?

  $JK^+ = \{ JKL\}$, yes, JK is super key, **1st condition for 3NF satisfied**

  Note JK is candidate key also (check yourself)

- Take $L \rightarrow K$ *is L* is super key ?

  $L^+ = \{ L\}$, no, L is not super key, **1st condition for 3NF NOT satisfied**

  K-L =K & K contained in candidate key (JK), **2nd condition for 3NF satisfied.**

"Therefore *R (J, K, L) is in 3NF*"

If you Replace J=s_ID ; K=dept_name & L= i_ID , this example is same as given in slide

Redundancy in 3NF

- There is some redundancy in this schema

  (the relation dept_advisor given in slide can also be discussed as below)

- **Example for problems due to redundancy in 3NF**
  - $R = (J, K, L)$
    $F = \{JK \rightarrow L, L \rightarrow K\}$

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

If you Replace J=s_ID ; K=dept_name & L= i_ID , this example is same as given in slide 75

- repetition of information (e.g., the relationship l_1, k_1)
  - i.e. (*i_ID, dept_name)* *in slide 75*
- need to use null values (e.g., to represent the relationship l_2, k_2 where there is no corresponding value for J).

(i.e. for an instructor in a department may not be an advisor for any student ,S_Id (i.e. J here is Null)

  - (*i_ID, dept_nameI)* if there is no separate relation mapping instructors to departments

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**

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 */

$R_j = R_i$;

$i=i-1$;

return $(R_1, R_2, ..., R_i)$

Note: algorithm ensures:

- each relation schema R_i is in 3NF
- decomposition is dependency preserving and lossless-join

3NF Decomposition example

Assume $R = (A, B, C)$

$F_c = \{A \rightarrow B\ ,\ B \rightarrow C\}$, **A is candidate key (if not given find yourself).**

***R* is not in 3 NF and Decompose to 3 NF**

Solution:

i=0

For loop

take $A \rightarrow B$, i=1 , R_1=A,B

take $B \rightarrow C$, i=2 , R_2 =B,C

IF none of R_1 ,R_2 contains Candidate key (i.e. A) is FALSE

because A is contained in R_1

Repeat

there is neither R_1 contained in R_2 nor R_2 contained in R_1

therefore No Deletion of duplicate schema

Return (R_1,R_2)

"R_1(A,B) & R_2(B,C) are decomposed 3 NF Relations"

3NF Decomposition: An Example

- Relation schema:

  cust_banker_branch = (*customer_id, employee_id, branch_name, type*)

  *Take customer_id, employee_id is the candidate key**

- The functional dependencies for this relation schema are:
  1. *customer_id, employee_id* → *branch_name, type*
  2. *employee_id* → *branch_name*
  3. *customer_id, branch_name* → *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* → *type*
employee_id → *branch_name*
customer_id, branch_name → *employee_id* }

* *Finding candidate key* *refer slide*

3NF Decompsition Example (Cont.)

- 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 namely- (*customer_id, branch_name, employee_id)*

  - 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)*

R(A,B,C,D,E,F,G,H) and F={ ABC →DE , E →BCG, F →AH }

Is it in 3NF Decompose into 3NF.

- Assume candidate keys are FE and FBC
- F_c ={ABC →DE , E →BCG, F →AH }
-

R(A,B,C,D,E,F,G,H,I) and F={ AB → C ,A → DE , B →F, F →GH, D →IJ}

Is it in 3NF Decompose into 3NF.

- Assume candidate keys are AB
- F_c={ AB → C ,A → DE , B →F, F →GH, D →IJ}

First Normal Form

- Domain is **atomic** if its elements are considered to be indivisible units
  - Examples of non-atomic domains:
    - Set of names, composite attributes
    - Identification numbers like CS101 that can be broken up into parts
- A relational schema R is in **first normal form** if the domains of all attributes of R are atomic
- Non-atomic values complicate storage of data
  - Example: Set of accounts stored with each customer, and set of owners stored with each account
  - We assume all relations are in first normal form (and revisit this in Object Based Databases)

First Normal Form (Cont'd)

- Atomicity is actually a property of how the elements of the domain are used.
  - Example: Strings would normally be considered indivisible
    - Suppose that students are given roll numbers which are strings of the form *CS0012* or *EE1127*
    - If the first two characters are extracted to find the department, the domain of roll numbers is not atomic.
  - Doing so is a bad idea: leads to encoding of information in application program rather than in the database.

Goal — Devise a Theory for the Following

- Decide whether a particular relation *R* is in "good" form.
- In the case that a relation *R* is not in "good" form, decompose it into a set of relations {R_1, R_2, ..., R_n} such that
  - each relation is in good form
  - the decomposition is a lossless-join decomposition
- Our theory is based on:
  - functional dependencies
  - multivalued dependencies

Functional Dependencies

- Functional Dependencies represents-Constraints on the set of **legal relations**.
- Require that the value for a certain set of attributes determines uniquely the value for another set of attributes.
- A functional dependency is a generalization of the notion of a *key*.
- Using the functional-dependency notation, ***K*** is some attribute/s of R
  - we say that ***K is a super key of r* (*R*)** if the functional dependency $K \rightarrow R$ holds on *r* (*R*).
- All the attributes of R are Functionally Depends on K **or** K functionally Determine all the attributes in R
- In other words, *K* is a super key if, for every legal instance of *r* (*R*), for every pair of tuples t_1 and t_2 from the instance, whenever $t_1[K] = t_2[K]$, it is also the case that $t_1[R] = t_2[R]$ (that is, $t_1 = t_2$).

Functional Dependencies (Cont.)

A	B	C	D
a_1	b_1	c_1	d_1
a_1	b_2	c_1	d_2
a_2	b_2	c_2	d_2
a_2	b_3	c_2	d_3
a_3	b_3	c_2	d_4

- Let *R* be a relation schema; **R** is **A,B,C,D**

$$\alpha \subseteq \boldsymbol{R} \text{ and } \beta \subseteq \boldsymbol{R}$$

- The **functional dependency**

$$\alpha \rightarrow \beta$$

holds on *R* if and only if for any legal relation two tuples t_1 **and** t_2 **of** ***r*** **agree on the attributes** α, they also **agree on the attributes** β. That is,

$$t_1[\alpha] = t_2[\alpha] \Rightarrow t_1[\beta] = t_2[\beta]$$

- Example: Consider *r*(A,*B*) with the following instance of *r.*

A	B
1	4
1	5
3	7

Observe that ***A*→*C*** is satisfied, but ***C*** → ***A*** is not satisfied*.

- On this instance, *A* → *B* does **NOT** hold,
- but *B* → *A* does hold.

* See the notes section

Functional dependency example

Assume that a college has many buildings such as Packard, Taylor, Watson and each building their class room numbers like 100,101,105 etc.. Every Class room in every building has some capacity. This information is represented in a relation Class_Room.

See the one of the legal instance of Class_room is below-

building	*room_number*	*capacity*
Packard	101	500
Painter	514	10
Taylor	3128	70
Watson	100	30
Watson	120	50

Note: Functional dependency is decided by meaning of attributes , not by any Instance of Relation

What are the FD that hold?

Building → Room_number

Building → Capacity

Room_number → Capacity

(Building,room_number) → Capacity

(Room_number ,Capacity) → Building

Room_number → Building

Capacity → Building

Capacity → Room_number

Capacity → (Building,room_number)

(Capacity ,Building) → Room_number

Following are some of the constraints that are expected to hold in a university database are:

1. Students and instructors are uniquely identified by their ID.
2. Each student and instructor has only one name.
3. Each instructor and student is (primarily) associated with only one department.
4. Each department has only one value for its budget, and only one associated building.

An instance of a relation that satisfies all such real-world constraints is called a **legal instance** of the relation; a legal instance of a database is one where all the relation instances are legal instances.

These constraints are represented using Functional Dependency concept.

i.e. **ID → STUDENT** **i.e student relation**

Name → STUDENT

ID → Name ; ID → Dept_Name

Dept_Name → Budget ; Dept_Name → Building

Do not expect the following to hold:

dept_name → salary ***because in real world salary is not determined by department ,so salary not Functionally depends on dept_name***

Summary of Last Class

- Randomly combining two relations or decomposing a relation will not give a good relation.
- A formal theory is needed which allows us to –
  - Decide a relation is in good form or bad form
  - If a relation is in Bad form, need to have method to decompose into sub relations which are in
    - ✓ Good form
    - ✓ Lossless Decomposition
  - Formal theory which supports above needs is-

    Functional Dependency.
    - ✓ Functional dependency is generalized notion of Key.
    - ✓ Key constraint can be expressed using functional dependency.
    - ✓ Functional dependency allows to express system constraints.

Assume **ITEMS(it_name, comp_name, price)** is a schema storing items(toothbrush, toothpaste etc.) and company(Colgate, Pepsodent etc.) names and price.

It_name	Comp_name	price
Toothpast	Colgate	20
Toothpast	Pepsodent	30
Toothbrush	Colgate	55
Toothbrush	Pepsodent	90

An It_name may be produced by many companies ,

therefore **It_name $\not\rightarrow$ Comp_Name**

($\not\rightarrow$ **means NOT -Functionally depends**)

A company may be producing many It_name,

therefore **Comp_Name $\not\rightarrow$ It_name**

Similarly price is not functionally depends on It_name or Comp_name

It_name $\not\rightarrow$ Price & Comp_Name $\not\rightarrow$ Price

However a Price is uniquely determined by the combination of (It_Name,Comp_Name), therefore

(It_Name,Comp_Name) $\rightarrow$ Price

Minimal Super Key

- ***K*** is a super key for relation schema ***R*** if and only if $K \rightarrow R$
- ***K*** is a candidate key(minimal super key) for *R* if and only if
  - $K \rightarrow R$, and
  - for no $\alpha \subset K$, $\alpha \rightarrow R$
- ***Example: In the ITEMS relation.***

Consider K means (It_Name,Comp_Name)

A Price is uniquely determined by the combination of (It_Name,Comp_Name), therefore

(It_Name,Comp_Name) ⟶ Price

Following two functional dependency holds because of trivial functional dependency (i.e. $\alpha \rightarrow \beta$ holds always if $\beta \subset \alpha$)

(It_Name,Comp_Name) → It_Name

(It_Name,Comp_Name) → Comp_Name

Therefore (It_Name,Comp_Name) ⟶ ITEMS

(It_Name,Comp_Name) is Super Key

As Discussed in previous slide **(It_name, Comp_name) is Super Key.**

- Further **K** i.e. **(It_name, Comp_name)** can be a **candidate key** (means minimal Super key) ,

  if **none of the subset of K *determine* R** i.e. **ITEMS.**

"None of the sub sets of (It_name, Comp_name) is Super key"

Subsets of (It_name, Comp_name) are

It_name , Comp_Name.

We know **It_name alone do not determine ITEMS &**

Comp_Name alone also do not determine ITEMS .

- **i.e It_name $\not\rightarrow$ ITEMS & Comp_name $\not\rightarrow$ ITEMS**

- From **(It_name, Comp_name)** we are unable to find any subset which **determine ITEMS .**

not possible to find any sub set of

(It_name, Comp_name) which is super key

- i.e. Hence **(It_name, Comp_name) are minimum attributes required to determine ITEMS**

- Therefore **(It_name, Comp_name) is minimal Super key (i.e. candidate key)**

Use of Functional Dependencies

- We **use functional dependencies** to:
  - **test relations** to see if they are legal under a given set of functional dependencies *F*.
    - If a relation *r* is legal under a set *F* of functional dependencies, we say that *r* **satisfies** *F.*
  - **specify constraints** on the set of legal relations*
    - We say that *F* **holds on** *R* if all legal relations on *R* satisfy the set of functional dependencies *F.*
- Note: A specific instance of a relation schema may satisfy a functional dependency even if the functional dependency does not hold on all legal instances.
  - For example, a specific instance of *instructor* may, by chance, satisfy

    name $\rightarrow$ *ID.*

**** See the notes section***

Trivial Functional Dependency

- *A* functional dependency is **trivial** if it is satisfied by all instances of a relation.
- **In general, $\alpha \rightarrow \beta$ is trivial if $\beta \subseteq \alpha$**
  - Example*:*
    - *(ID, name)* $\rightarrow$ *ID is trivial because ID* $\subseteq$ *(ID, name)*
    - *name* $\rightarrow$ *name*

Write the Functional Dependencies for a given System Requirements

- Assume that we want to store information about employees and department in which they are working.
- Every employee has a unique EmpNo. About each employee we are interested to store his name ,salary what he earns, city in which he resides , pincode , STDCode of the city and PhoneNumber of the employee. Every city has one pincode (ignoring the point that-with in a city different areas will have different Pincode) , similar type assumption we considered here like every city has one STDCode.
- Each employee works exactly in one department. More than one employees work in a department. About each department we want to record department number-Deptno and department name Dname, Department location –Loc. Every department has a unique deptno, every department has one unique Dname and each department is situated in one Location.

See sample data corresponding to these constraints in next slide

Sample data corresponding to the constraints in previous slide

Empno	Ename	Sal	City	PinCode	STD_Code	Phone_Number	Deptno	Dname	Loc
157	Niraj	1000	MANIPAL	576104	820	12345	D1	ADMINSTION	MANGALORE
100	Ravi	2000	MANGALORE	587693	819	56788	D2	MARKETING	BOMBAY
102	Raviraj	3899	BANGLAORE	580001	80	786536	D3	RESEARCH	BOMBAY
111	Raghu	5788	MANIPAL	576104	820	125978	D1	ADMINSTION	MANGALORE
150	X	2467	BOMBAY	489921	420	678899	D2	MARKETING	BOMBAY
103	Y	9763	HYDERBAD	765690	560	567899	D3	RESEARCH	BOMBAY
109	Z	46789	CHENNAI	674321	730	656890	D1	ADMINSTION	MANGALORE
125	Manu	5788	BANGLAORE	580001	80	123456	D3	RESEARCH	BOMBAY
104	A	8653	MANIPAL	576104	820	89865	D2	MARKETING	BOMBAY
106	B	79076	CHENNAI	674321	730	566778	D3	RESEARCH	BOMBAY
123	Mahesh	2000	MANGALORE	587693	819	55677	D2	MARKETING	BOMBAY
108	Ravi	44466	CHENNAI	674321	730	57780	D4	PRODUCTION	MANGALORE
124	D	44467	HYDERBAD	765690	560	786535	D5	PURCHASE	CHENNAI
113	Z	65576	MANIPAL	576104	820	89627	D5	PURCHASE	CHENNAI

Set of Functional Dependencies that hold

Following are the some FDs that hold on the system as per the requirements given in slide 21-22

Empno → Name; Empno → Sal ; Empno → Deptno;

Empno → Dname; Empno → City; **Empno → PinCode;**

Empno → STD_Code; *Empno* →Phone_Number; *Empno* →Loc

Deptno→ Dname; Deptno→ Loc; Dname → Deptno;

Dname → Loc; City →PinCode; PinCode → City;

City → STD_Code; STD_Code → *City;* PinCode →STD_Code

Whether these Functional Dependency Hold ?

Deptno → Ename; Deptno → Sal; Dname → Sal;

City → Dname; City →Deptno; Phone_Number → STD_Code;

City→PhoneNumber.

IS (Empno , Name) → Deptno ;

Closure of a Set of Functional Dependencies

- Given a set *F* of functional dependencies, there are certain other functional dependencies that are logically implied by *F*.
  - **For example:**
  - If $A \rightarrow B$ **and** $B \rightarrow C$, then we can infer that $A \rightarrow C$
- The set of **all** functional dependencies logically implied by *F* is the **closure** of ***F***.
- We denote the ***closure* of *F*** by F^+.
- F^+ is a superset of ***F***

Armstrong's Axioms

- We can find **F^+**, the closure of F, by repeatedly applying **Armstrong's Axioms:**
  - if $\beta \subseteq \alpha$, then $\alpha \rightarrow \beta$ **(reflexivity)**
  - if $\alpha \rightarrow \beta$, then $\gamma\ \alpha \rightarrow \gamma\ \beta$ **(augmentation)**
  - if $\alpha \rightarrow \beta$, and $\beta \rightarrow \gamma$, then $\alpha \rightarrow \gamma$ **(transitivity)**
- These rules are
  - **sound** (do **not generate any incorrect** functional dependencies), and
  - **complete** (generate **all functional** dependencies that hold).
  - There are *additional rules which can be proved based on these Armstrong Rules see slide 25.

*** see the proof in notes section below in slide 25**

Example

Proving some logically implied set of functional dependencies by applying Armstrong rules

- $R = (A, B, C, G, H, I)$
  $F = \{ A \rightarrow B$
  $A \rightarrow C$
  $CG \rightarrow H$
  $CG \rightarrow I$
  $B \rightarrow H \}$
- some members of F^+
  - $A \rightarrow H$
    - by transitivity from $A \rightarrow B$ *and* $B \rightarrow H$
  - $AG \rightarrow I$
    - by augmenting $A \rightarrow C$ with G, to get $AG \rightarrow CG$ and then transitivity with $CG \rightarrow I$
  - $CG \rightarrow HI$
    - by augmenting $CG \rightarrow I$ *with* CG to infer $CG \rightarrow CGI$,
      and augmenting of $CG \rightarrow H$ *with* I to infer $CGI \rightarrow HI$, and then transitivity

if $\beta \subseteq \alpha$, **then** $\alpha \rightarrow \beta$
(reflexivity)

if $\alpha \rightarrow \beta$, **then** $\gamma\ \alpha \rightarrow \gamma\ \beta$
(augmentation)

if $\alpha \rightarrow \beta$, **and** $\beta \rightarrow \gamma$, **then** $\alpha \rightarrow \gamma$
(transitivity)

Some Additional rules

- *** Additional rules:**
  - If $\alpha \rightarrow \beta$ holds *a*nd $\alpha \rightarrow \gamma$ holds, then $\alpha \rightarrow \beta\gamma$ holds (**union**)
  - If $\alpha \rightarrow \beta\ \gamma$ holds, then $\alpha \rightarrow \beta$ holds and $\alpha \rightarrow \gamma$ holds (**decomposition**)
  - If $\alpha \rightarrow \beta$ holds *a*nd $\gamma\ \beta \rightarrow \delta$ holds, then $\alpha\ \gamma \rightarrow \delta$ holds (**pseudo transitivity**)

The above rules can be **inferred from Armstrong's axioms**.

* see the proof in notes section below

Procedure for Computing F+

- To compute the closure of a set of functional dependencies F:

$F^+ = F$

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

      then add the resulting functional dependency to F^+

until F^+ does not change any further

NOTE: We shall see an alternative procedure for this task later

Closure of Attribute Sets

- Given a set of attributes a, define the ***closure*** of **a under *F*** (denoted by a^+) as the **set of attributes that are functionally determined by a under *F***

Algorithm **to compute a^+,** the closure of a under *F*

```
result := a;
while (changes to result) do
    for each β → γ in F do
        begin
            if β ⊆ result then result := result ∪ γ
        end
```

Example for Attribute Set Closure

- $R = (A, B, C, G, H, I)$
- $F = \{ A \rightarrow B$
  $A \rightarrow C$
  $CG \rightarrow H$
  $CG \rightarrow I$
  $B \rightarrow H \}$
- $(AG)^+$
  1. *result* = AG
  2. *result* = $ABCG$ ($A \rightarrow C$ and $A \rightarrow B$)
  3. *result* = $ABCGH$ ($CG \rightarrow H$ and $CG \subseteq AGBC$)
  4. *result* = $ABCGHI$ ($CG \rightarrow I$ and $CG \subseteq AGBCH$)
- Is AG a candidate key? *
  1. Is AG a super key?
     1. Does $AG \rightarrow R$? == Is $(AG)^+ \supseteq R$
  2. Is any subset of AG a super key?
     1. Does $A \rightarrow R$? == Is $(A)^+$ contains R
     2. Does $G \rightarrow R$? == Is $(G)^+$ contains R
  2. Is A is super key?

* See slide 17,18 for testing – is the super key is also candidate key ?

Step-by-step approach to Find AG^+

1. *result = AG*
2. *While loop* (*1st While loop*)
3. *For Loop*

1. take $A \rightarrow B$

$A \subseteq$ result

result = result $\cup$ *B ={ AGB}*

2. take $A \rightarrow C$

$A \subseteq$ result

result = result $\cup$ *C ={ AGBC}*

3. Take $CG \rightarrow H$

$CG \subseteq$ result

result = result $\cup$ *H ={ AGBCH}*

4. Take $CG \rightarrow I$

$CG \subseteq$ result

result = result $\cup$ *I ={ AGBCHI}*

5. Take $B \rightarrow H$

$B \subseteq$ result

result = result $\cup$ *H ={ AGBCHI}*

End of For Loop

….Step-by-step approach to Find AG^+

At the end of 1st While loop result= { AGBCHI}
So when 2nd While loop starts result= { AGBCHI}

1. *While loop (2nd While loop)*
2. *For Loop*
   1. take $A \rightarrow B$
      $A \subseteq$ result
      result = result $\cup$ *B ==*{ AGBCHI}
   2. take $A \rightarrow C$
      $A \subseteq$ result
      result = result $\cup$ *C ==*{ AGBCHI}
   3. Take $CG \rightarrow H$
      $CG \subseteq$ result
      result = result $\cup$ *H =*{ AGBCHI}
   4. Take $CG \rightarrow I$
      $CG \subseteq$ result
      result = result $\cup$ *I =*{ AGBCHI}
   5. *Take* $B \rightarrow H$
      $B \subseteq$ result
      result = result $\cup$ *H =*{ AGBCHI}

   End of For Loop

Result from While loop 1 and While loop 2 are same *, there is* ***no change in result*** *, therefore* ***exit While loop***

Exercises

1. *R (A, B, C, D)*

   $F = \{ A \rightarrow B \ , B \rightarrow C \ , \quad B \rightarrow D \ \}$, Find A^+ , B^+

2. *R (A,B,C,D)*

   $F = \{C \rightarrow D \ , A \rightarrow B \ , B \rightarrow C \ \}$ Find A^+ , C^+

Uses of Attribute Closure

There are several uses of the attribute closure algorithm:

- **Testing for superkey:**
  - To test if α is a super key*,
    - **we compute α^+, and check if α^+ contains all attributes of *R*.**
- **Testing functional dependencies**
  - To check if a functional dependency **$\alpha \rightarrow \beta$ holds** (or, in other words, is in F^+), **just check if $\beta \subseteq \alpha^+$.**
  - That is, we **compute α^+** by using attribute closure, and then check **if it contains β**.
  - Is a simple and cheap test, and very useful
- **Computing closure of F**
  - For each $\gamma \subseteq R$, we find the closure γ^+, and for each $S \subseteq \gamma^+$, we output a functional dependency $\gamma \rightarrow S$.

(see Notes section for explanation)

* *See slide 17,18 for testing – is the super key is also candidate key ?*

Example for Testing for super key

$R = (A, B, C, G, H, I)$

$F = \{ \quad A \rightarrow B \ , A \rightarrow C \ , CG \rightarrow H \ , CG \rightarrow I \ , \quad B \rightarrow H \ \}$

IS AG is Super Key ?

Solution:

- Find **AG^+**
- From slide 30 we know AG^+ is

  $AG^+ = \{ABCGHI\}$

  means ***$AG \rightarrow ABCGHI$*** *i.e.* ***$AG \rightarrow$ R***

  (i.e. AG functionally determine all the attributes of R)

If *$AG \rightarrow$* R then AG is super key of R

Is AG is minimal Super key ?

(use the idea –

K is a candidate key(minimal super key) for *R* if and only if

- ***$K \rightarrow R$,*** and
- **for no $\alpha \subset K$, $\alpha \rightarrow R$)**
- See next slide

Example for Checking a super key is minimal super key or not

From previous slide we know AG is super key of R.

Find Sub sets of AG- A and G

Find A^+ and G^+

IF $A \rightarrow R$ (A is super key) OR $G \rightarrow R$ (G is Super key) Then

AG is NOT Minimal Super Key, but only Super Key

ELSE

AG is also Minimal Super Key

$A^+ = \{ ABCH \}$

A $\not\rightarrow$ R i.e. A is not Super key for R

$G^+ = \{ G \}$

G $\not\rightarrow$ R i.e. G is not Super key for R

Subsets of AG , neither A nor G is Super key of relation R

Therefore AG is minimal Super Key (Candidate Key)

Example for Testing functional dependencies

To check if a functional dependency $\alpha \rightarrow \beta$ holds (or, in other words, is in F^+), just check if $\beta \subseteq (\notin$ If β $\alpha^+ \not\rightarrow$ n α β

R = (A, B, C, G, H, I)

***F* = { $A \rightarrow B$, $A \rightarrow C$, $CG \rightarrow H$, $CG \rightarrow I$, $B \rightarrow H$ }**

Is $CG \rightarrow A$ holds ?

Solution:

- Find **CG^+**
- IF **$A \in CG^+$** Then

  $CG \rightarrow A$ holds

  ELSE

  CG $\not\rightarrow$ A (means functional dependency do not hold)

Exercise:

Is $AC \rightarrow H$ holds ?
Is $AG \rightarrow I$ holds ?
Is $AI \rightarrow G$ holds ?

Example for Computing F^+

For each $\gamma \subseteq R$, we find the closure γ^+, and for each $S \subseteq \gamma^+$, we output a functional dependency $\gamma \rightarrow S$.

$R = (A, B, C)$

$F = \{ A \rightarrow B \ , B \rightarrow C \}$

All the subsets of R are –A,B,C,AB,AC,BC,ABC

Take one subset from above **set say -A** and **find A^+**

A^+ ={A ,B,C}

Subsets of A^+ are - A,B,C,AB,AC,BC,ABC

Therefore following functional dependencies hold

$A \rightarrow B$, $A \rightarrow C$, $A \rightarrow AB$, $A \rightarrow AC$, $A \rightarrow BC$, $A \rightarrow ABC$. ----

(1)Similarly , **take next subset say B** and **Find B^+**

B^+={B,C}

Subsets of B^+ are - B,C,BC

Therefore following functional dependencies hold

B →C ,B →BC ----(2)

Take subset C , Find C^+

C^+ ={ C}

Therefore **C →C (similarly A →A and B →B holds)** holds ----(3)

….Example for Computing F^+

Take AB

AB^+= (A,B,C}

Subsets of AB+ are - A,B,C,AB,AC,BC,ABC

Therefore following functional dependencies hold

AB → B , AB → C , A B→ AB, AB → AC, A B→ BC, AB → ABC ----(4)

Take BC

BC^+ ={ B,C}

Subsets of BC+ are - B,C,BC

Therefore following functional dependencies hold

BC →B,BC →C,BC →BC ---- (5)

Take AC

AC^+= {A,B,C}

Subsets of AC^+ are - A,B,C,AB,AC,BC,ABC

Therefore following functional dependencies hold

AC → B , AC → C , AC→ AB, AC → AC, AC→ BC, AC → ABC. ----(6)

….Example for Computing F^+

Take ABC

ABC+={A,B,C}

Subsets of ABC^+ are - A,B,C,AB,AC,BC,ABC

Therefore following functional dependencies hold

$\mathbf{ABC \rightarrow B}$, $\mathbf{ABC \rightarrow C}$, $\mathbf{ABC \rightarrow AB}$, $\mathbf{ABC \rightarrow AC}$, $\mathbf{ABC \rightarrow BC}$, $\mathbf{ABC \rightarrow ABC}$. ----(7)

From (1) (2) (3) (4) (5) (6) (7)

by eliminating duplicates we can get all functional dependencies set

$\mathbf{F^+ = \{}$ $A \rightarrow A$, $B \rightarrow B$, $C \rightarrow C$, $A \rightarrow B$, $A \rightarrow C$, $A \rightarrow AB$, $A \rightarrow AC$, $A \rightarrow BC$,
$A \rightarrow ABC$, $B \rightarrow C$, $B \rightarrow BC$, $AB \rightarrow B$, $AB \rightarrow C$, $AB \rightarrow AB$, $AB \rightarrow AC$,
$AB \rightarrow BC$, $AB \rightarrow ABC$, $AC \rightarrow B$, $AC \rightarrow C$, $AC \rightarrow AB$, $AC \rightarrow AC$,
$AC \rightarrow BC$, $AC \rightarrow ABC$, $ABC \rightarrow B$, $ABC \rightarrow C$, $ABC \rightarrow AB$, $ABC \rightarrow AC$,
$ABC \rightarrow BC$, $ABC \rightarrow ABC$ $\mathbf{\}}$

Canonical Cover

- Sets of functional dependencies may have redundant dependencies that can be inferred from the others
  - For example: $A \rightarrow C$ is redundant in: $\{A \rightarrow B, \quad B \rightarrow C, A \rightarrow C\}$
  - Parts of a functional dependency may be redundant
    - E.g.: on RHS: $\{A \rightarrow B, \quad B \rightarrow C, \quad A \rightarrow CD\}$ can be simplified to

      $\{A \rightarrow B, \quad B \rightarrow C, \quad A \rightarrow D\}$
    - E.g.: on LHS: $\{A \rightarrow B, \quad B \rightarrow C, \quad AC \rightarrow D\}$ can be simplified to

      $\{A \rightarrow B, \quad B \rightarrow C, \quad A \rightarrow D\}$
- Intuitively, a canonical cover of F is a "minimal" set of functional dependencies equivalent to F, having no redundant dependencies or redundant parts of dependencies

Extraneous Attributes

- Consider a set F of functional dependencies and the functional dependency $\alpha \rightarrow \beta$ in F.
  - Attribute A is **extraneous** in α if $A \in \alpha$ (an attribute A on the left side of $\alpha \rightarrow \beta$) and F logically implies $(F - \{\alpha \rightarrow \beta\}) \cup \{(\alpha - A) \rightarrow \beta\}$.
  - Attribute A is **extraneous** in β if $A \in \beta$ (an attribute A on the right side of $\alpha \rightarrow \beta$) and the set of functional dependencies $(F - \{\alpha \rightarrow \beta\}) \cup \{\alpha \rightarrow (\beta - A)\}$ logically implies F.

- Example: Given $F = \{A \rightarrow C, AB \rightarrow C\}$
  - B is extraneous in $AB \rightarrow C$ because $\{A \rightarrow C, AB \rightarrow C\}$ logically implies $A \rightarrow C$ (I.e. the result of dropping B from $AB \rightarrow C$).
- Example: Given $F = \{A \rightarrow C, AB \rightarrow CD\}$
  - C is extraneous in $AB \rightarrow CD$ since $AB \rightarrow C$ can be inferred even after deleting C

Testing if an Attribute is Extraneous

- Consider a set *F* of functional dependencies and the functional dependency $\alpha \rightarrow \beta$ in *F*.
- To test if attribute $A \in \alpha$ is extraneous in α
  1. compute $(\{\alpha\} - A)^+$ using the dependencies in *F*
  2. check that $(\{\alpha\} - A)^+$ contains β; if it does, *A* is extraneous in α
- To test if attribute $A \in \beta$ is extraneous in β
  1. compute α^+ using only the dependencies in
     $$F' = (F - \{\alpha \rightarrow \beta\}) \cup \{\alpha \rightarrow (\beta - A)\},$$
  2. check that α^+ contains *A;* if it does, *A* is extraneous in β

Canonical Cover

- A **canonical cover** for F is a set of dependencies F_c such that
  - F logically implies all dependencies in F_c, and
  - F_c logically implies all dependencies in F, and
  - No functional dependency in F_c contains an extraneous attribute, and
  - Each left side of functional dependency in F_c is unique.
- To compute a canonical cover for F:
- $F_c = F$

  repeat

  Use the union rule to replace any dependencies in F_c

  $\alpha_1 \rightarrow \beta_1$ 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: test for extraneous attributes done using F_c, not F */

  If an extraneous attribute is found, delete it from $\alpha \rightarrow \beta$

  until F does not change
- Note: Union rule may become applicable after some extraneous attributes have been deleted, so it has to be re-applied

Computing a Canonical Cover

- $R = (A, B, C)$
  $F = \{A \rightarrow BC$
  $B \rightarrow C$
  $A \rightarrow B$
  $AB \rightarrow C\}$
- Combine $A \rightarrow BC$ and $A \rightarrow B$ into $A \rightarrow BC$
  - Set is now $\{A \rightarrow BC, B \rightarrow C, AB \rightarrow C\}$
- A is extraneous in $AB \rightarrow C$
  - Check if the result of deleting A from $AB \rightarrow C$ is implied by the other dependencies
    - Yes: in fact, $B \rightarrow C$ is already present!
  - Set is now $\{A \rightarrow BC, B \rightarrow C\}$
- C is extraneous in $A \rightarrow BC$
  - Check if $A \rightarrow C$ is logically implied by $A \rightarrow B$ and the other dependencies
    - Yes*:* using transitivity on $A \rightarrow B$ *and* $B \rightarrow$ C.
      - Can use attribute closure of A in more complex cases
- The canonical cover is: $A \rightarrow B$
  $B \rightarrow C$

Lossless-join Decomposition

- For the case of **R = (R_1, R_2)**, we require that for all possible relations *r* on schema *R*

$$r = \prod_{R1}(r) \bowtie \prod_{R2}(r)$$

- A decomposition of *R* into R_1 and R_2 is lossless join if at least one of the following dependencies is in F^+:
  - $R_1 \cap R_2 \rightarrow R_1$
  - $R_1 \cap R_2 \rightarrow R_2$

This implies common attribute of R1 and R2 must be super key of R_1 or R_2

- The above functional dependencies are a sufficient condition for lossless join decomposition; the dependencies are a necessary condition only if all constraints are functional dependencies

Example

- $R = (A, B, C)$
  $F = \{A \rightarrow B, B \rightarrow C)$
  - Can be decomposed in two different ways
- $\boldsymbol{R_1 = (A, B), \quad R_2 = (B, C)}$
  - Lossless-join decomposition:

    $$\boldsymbol{R_1 \cap R_2 = \{B\} \text{ and } B \rightarrow BC}$$

  - Dependency preserving
- $\boldsymbol{R_1 = (A, B), \quad R_2 = (A, C)}$
  - Lossless-join decomposition:

    $$\boldsymbol{R_1 \cap R_2 = \{A\} \text{ and } A \rightarrow AB}$$

  - Not dependency preserving
    (cannot check $B \rightarrow C$ without computing $R_1 \bowtie R_2$)

Dependency Preservation

- Let F_i be the set of dependencies F^+ that include only attributes in R_i.
  - A decomposition is **dependency preserving**, if

    $(F_1 \cup F_2 \cup \ldots \cup F_n)^+ = F^+$
  - If it is not, then checking updates for violation of functional dependencies may require computing joins, which is expensive.

Testing for Dependency Preservation

- To check if a dependency $\alpha \rightarrow \beta$ is preserved in a decomposition of ***R*** **into** $R_1, R_2, \ldots, R_n$ we apply the following test (with attribute closure done with respect to *F*)

  result = α
  while (changes to *result*) do
  for each R_i in the decomposition
  $t = (result \cap R_i)^+ \cap R_i$
  $result = result \cup t$

  - If ***result*** **contains all attributes in** β, then the functional dependency $\alpha \rightarrow \beta$ **is preserved**.

- We apply the test on all dependencies in *F* to check if a decomposition is dependency preserving
- This procedure **takes polynomial time**, instead of the exponential time required to compute F^+ **and** $(F_1 \cup F_2 \cup \ldots \cup F_n)^+$

Example using Algorithm

- Given the following:

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

  F = {AB→C, C→E, B→D, E→A}

  Assume R is Decomposed into R_1 and R_2 as below-

  R_1(B,C,D) R_2(A,C,E)

- **Is this decomposition dependency preserving?**

Example continued…

R(A,B,C,D,E) F = {AB→C, C→E, B→D, E→A}

Decomposition: R1(B,C,D) R2(A,C,E)

Is AB→C Preserved?

Result=AB

For Result ∩ R1 = AB ∩ BCD = B

$\{B\}^+ = BD$

$\{B\}^+ \cap R1 = BD \cap BCD = BD$

Update Result = AB ∪ BD = ABD, continue

result = α

while (changes to *result*) do

for each R_i in the decomposition

$t = (result \cap R_i)^+ \cap R_i$

$result = result \cup t$

Example continued…

R(A,B,C,D,E)F = {AB→C, C→E, B→D, E→A}

Decomposition: R1(B,C,D) R2(A,C,E)

Result=ABD

For Result ∩ R2 = ABD ∩ ACE = A

$\{A\}^+ = A$

$\{A\}^+ \cap R2 = A \cap ACE = A$

Update Result, Result is still ABD

Since Result changed, repeat checking R1 to R2.

Example continued…

R(A,B,C,D,E)F = {AB→C, C→E, B→D, E→A}

Decomposition: R1(B,C,D) R2(A,C,E)

Result=ABD

For Result ∩ R1 = ABD ∩ BCD = BD

$\{BD\}^+ = BD$

$\{BD\}^+ \cap R1 = BD \cap BCD = BD$

Update Result = ABD ∪ BD = ABD, so Result hasn't changed but you still have to continue.

Example

R(A,B,C,D,E)F = {AB→C, C→E, B→D, E→A}

Decomposition: R1(B,C,D) R2(A,C,E)

Result=ABD

For Result ∩ R2 = ABD ∩ ACE = A

$\{A\}^+ = A$

$\{A\}^+ \cap R2 = A \cap ACE = A$

Update Result, Result is still ABD

Since Result hasn't change, we stop here.

Check, Is Result Contains C ? **NO**

Therefore we can conclude

AB→C is not preserved.

Similarly check other functional dependencies.

Example continued…

R(A,B,C,D,E)F = {AB→C, C→E, B→D, E→A}

Decomposition: R1(B,C,D) R2(A,C,E)

Is B→D Preserved ?

Result=B

For Result ∩ R1 = B ∩ BCD = B

$\{B\}^+ = BD$

$\{B\}^+ \cap R1 = BD \cap BCD = BD$

Update Result = B ∪ BD = BD

Since D belongs to Result,

B→D is preserved.

Example continued…

R(A,B,C,D,E)F = {AB→C, C→E, B→D, E→A}

Decomposition: R1(B,C,D) R2(A,C,E)

Is C→E Preserved ?

Result=C

For Result ∩ R2 = C ∩ ACE = C

$\{C\}^+ = CEA$

$\{C\}^+ \cap R1 = CEA \cap ACE = ACE$

Update Result = C ∪ACE= ACE

Since E is in Result i.e ACE,

C→E is preserved.

Example continued…

R(A,B,C,D,E)F = {AB→C, C→E, B→D, E→A}

Decomposition: R1(B,C,D) R2(A,C,E)

Is E→A Preserved ?

Result=E

For Result ∩ R1 = E ∩ ACE = E

$\{E\}^+ = EA$

$\{E\}^+ \cap R1 = EA \cap ACE = EA$

Update Result = E ∪ EA= EA

Since A belongs to Result i.e. EA,

E→A is preserved.

As one of the functional dependencies AB→C is not is not preserved, Therefore decomposition R1,R2 is not dependency preserving decomposition.

Example

R(A,B,C,D,E) F = {AB→C, C→E, B→D, E→A}

Decomposition: R1(B,C,D) R2(A,C,E)

Shortcut:

For any functional dependency, if both LHS and RHS collectively are within any of the sub scheme R_i. Then this functional dependency is preserved.

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 trivial (i.e., $\beta \subseteq \alpha$)
- α is a super key for *R*

Example schema *not* in BCNF:

instr_dept (*ID, name, salary, dept_name, building, budget*)

Assume F={ *ID* → *name, salary , dept_name, dept_name* → *building, budget* }

instr_dept is not in BCNF because *dept_name*→ (*building, budget*) holds on ***instr_dept***, but ***dept_name* is not a superkey**

Decomposing a Schema into BCNF

- Suppose we have a schema *R* and a non-trivial dependency $\alpha \rightarrow \beta$ causes a violation of BCNF.

  We decompose *R* into:

  - $(\alpha \cup \beta)$
  - $(R - (\beta - \alpha))$

On decomposing a Relation R ,one or more of the resulting schemas may not be in BCNF. In such cases, further decomposition is required, the eventual result of which is a set of BCNF schemas.

- In our example,
- **Inst_dept (ID, Name, Salary, Dept_Name, Building, Budget)**
- **Assume *dept_name →building, budget*** *violates*
  - α = *dept_name* β = *building, budget*

    and ***inst_dept*** is replaced by
  - $(\alpha \cup \beta)$ = (*dept_name, building, budget*)
  - (*R* - (β - α)) = (*ID, name, salary, dept_name*)
  - **R_1(*dept_name, building, budget*)**
  - **R_2(*ID, name, salary, dept_name*)**

BCNF and Dependency Preservation

- Constraints*, including functional dependencies, are costly to check in practice unless they pertain to only one relation
- If it is sufficient to test only those dependencies on each individual relation of a decomposition in order to ensure that *all* functional dependencies hold, then that decomposition is *dependency preserving.*
- Because it is not always possible to achieve both BCNF and dependency preservation, we consider a weaker normal form, known as *third normal form.*
  - suppose that we make a small change to our university Organization, see the ER diagram in next slide.

    The "small" change we shall make is that an instructor can be associated with only a single department and a student may have more than one advisor, but at most one from a given department.

** See the notes section*

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,**

     α is a super key of *R*.
  - It suffices to check only the dependencies in the given set F for violation of BCNF, rather than check all dependencies in $\mathbf{F}^+$.
  - We can show that if none of the dependencies in F causes a violation of BCNF, then none of the dependencies in $\mathbf{F}^+$ will cause a violation of BCNF,either.
- However, procedure does not work **when a relation is decomposed**.
- That is, **it *does not* suffice** to use *F* when we test a relation R_i , in a decomposition of *R*, for violation of BCNF.

Example

- $R = (A, B, C)$
  $F = \{A \rightarrow B,\ B \rightarrow C\}$
  Key = $\{A\}$
- R is not in BCNF (why?)
- Assume R is Decomposed into $R_1 = (A, B)$, $R_2 = (B, C)$
  - R_1 and R_2 in BCNF (check*) (see notes section)s
  - Lossless-join decomposition (check) (slide 47 example)
  - Dependency preserving (check)

* Using F it is not possible to check whether R1 and R2 are in BCNF. We need functional dependency set F1,F2 corresponding to R1 and R2 respectively.

Testing Decomposition for BCNF

when R is decomposed into R_i

- To check if a relation R_i in a decomposition of R is in BCNF,
  - Either test 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: (**this method we use in further discussion**)
    - 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.

Example: $R = (A, B, C)$

$F = \{A \rightarrow B,\ B \rightarrow C\}$,

R is not in BCNF and we got Decomposition $R_1 = (A, B)$, $R_2 = (B, C)$

Is R_1 and R_2 in BCNF ?

Example: $R = (A, B, C)$

$F = \{A \rightarrow B,\ B \rightarrow C\}$,

R is not in BCNF and we got Decomposition

$R_1 = (A, B)$, $R_2 = (B, C)$ Is R_1 and R_2 in BCNF ?

Take R_1 all subsets of R_1 are A , B ,AB

1. $A^+ = \{A,B,C\}$ $R_1 - A = B$

A^+ do not include any attribute of **$R_1 - A$ is FALSE**, but

A^+ include ALL attributes of R_1 is TRUE

2. $B^+ = \{B,C\}$ $R_1 - B = A$

B^+ do not include any attribute of **$R_1 - B$ is TRUE**

R_1 is in BCNF

Take R_2 all subsets of R_2 are B,C ,BC

1. $B^+ = \{B,C\}$ $R_2 - B = C$

B^+ do not include any attribute of **$R_2 - B$ is FALSE**, but

B^+ include ALL attributes of R_2 is TRUE

2. $C^+ = \{C\}$

C^+ do not include any attribute of **$R_2 - C$ is TRUE**

R_2 is in BCNF

BCNF Decomposition Algorithm

A **general method to decompose** a relation schema so as to satisfy BCNF.

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** $\alpha \cap \beta = \varnothing$
          ***result* := (*result* – R_i) $\cup$ (R_i – β) $\cup$ (α, β) ;**
      end
    else *done* := **true ;**

The decomposition that this algorithm generates is **not only in BCNF**, but is also a **lossless decomposition**.

BCNF decomposition Example

Assume $R = (A, B, C)$

$F = \{A \rightarrow B,\ B \rightarrow C\}$,

R is not in BCNF and Decompose to BCNF

Initially as there are result= {R} & Done:=False

Calculate F^+ refer slide 40 for calculating F^+

$F^+ = \{ A \rightarrow A, B \rightarrow B, C \rightarrow C, A \rightarrow B, A \rightarrow C, A \rightarrow AB, A \rightarrow AC, A \rightarrow BC,$
$A \rightarrow ABC, B \rightarrow C, B \rightarrow BC, AB \rightarrow B, AB \rightarrow C, AB \rightarrow AB, AB \rightarrow AC, AB \rightarrow BC,$
$AB \rightarrow ABC, AC \rightarrow B, AC \rightarrow C, AC \rightarrow AB, AC \rightarrow AC, AC \rightarrow BC, AC \rightarrow ABC,$
$ABC \rightarrow B, ABC \rightarrow C, ABC \rightarrow AB, ABC \rightarrow AC, ABC \rightarrow BC, ABC \rightarrow ABC \}$

Non-Trivial FD from F+

$F_1^+ = \{A \rightarrow B, A \rightarrow C, A \rightarrow AB, A \rightarrow AC, A \rightarrow BC, A \rightarrow ABC,$
$B \rightarrow C, B \rightarrow BC, AB \rightarrow C, AB \rightarrow AC, AB \rightarrow BC, AB \rightarrow ABC, AC \rightarrow B, AC \rightarrow$
$AB, AC \rightarrow BC, AC \rightarrow ABC \}$

…BCNF decomposition Example

$\alpha \rightarrow R_i$ is not in F^+

In F_1^+ find of functional dependencies of type $\alpha \rightarrow R_i$ *means where* α is super key for the decomposition R_i , *in fact in this example yet (in the beginning) R is not decomposed ,therefore R is taken.*

Find all functional dependencies in which *LHS (i.e.* α) is super key and remove all such functional dependencies from F_1^+

In this example A is the key and hence any attribute in combination with A will be super keys.

Therefore we can remove $A \rightarrow B$, $A \rightarrow C$, $A \rightarrow AB$, $A \rightarrow AC$, $A \rightarrow BC$, $A \rightarrow ABC$, $AB \rightarrow C$,$AB \rightarrow AC$, $AB \rightarrow BC$, $AB \rightarrow ABC$, $AC \rightarrow B$, $AC \rightarrow AB$, $AC \rightarrow BC$, $AC \rightarrow ABC$ from F_1^+ and let us call as F_2^+ **(see next slide)**

…BCNF decomposition Example

$F_2^+ = \{B \rightarrow C, B \rightarrow BC\}$

In these functional dependencies (F_2^+) take functional dependency $\alpha \rightarrow \beta$ **such that** $\alpha \cap \beta = \varnothing$

From F_2^+

for $B \rightarrow C$ $B \cap C = \varnothing$

for $B \rightarrow BC$ $B \cap BC \neq \varnothing$

Therefore only $B \rightarrow C$ **is used in the algorithm to test**

$$result := (result - R_i) \cup (R_i - \beta) \cup (\alpha, \beta) \ ;$$

In the beginning as there are no decompositions (R_i) , result={R} itself.

$$result := (R - R) \cup (R - C) \cup (B,C)$$

$$result := \varnothing \cup (A,B) \cup (B,C)$$

Now result is two decomposed relations as below-

result:= { (A,B) , (B,C)}

Therefore R_1 (A,B) , R_2(B,C)

In the next while loop , check R_1 for BCNF , it is in BCNF , so retain as it is.

In next while loop ,check R_2 for BCNF , it is also in BCNF , so retain as it is.

Finally in the ELSE part done:= TRUE and exits from algorithm

Decomposition into BCNF (SIMPLE APPROACH)

Given A Relation R with FD's F

- Look among the given FD's for a BCNF violation $X \rightarrow Y$
- Compute X^+

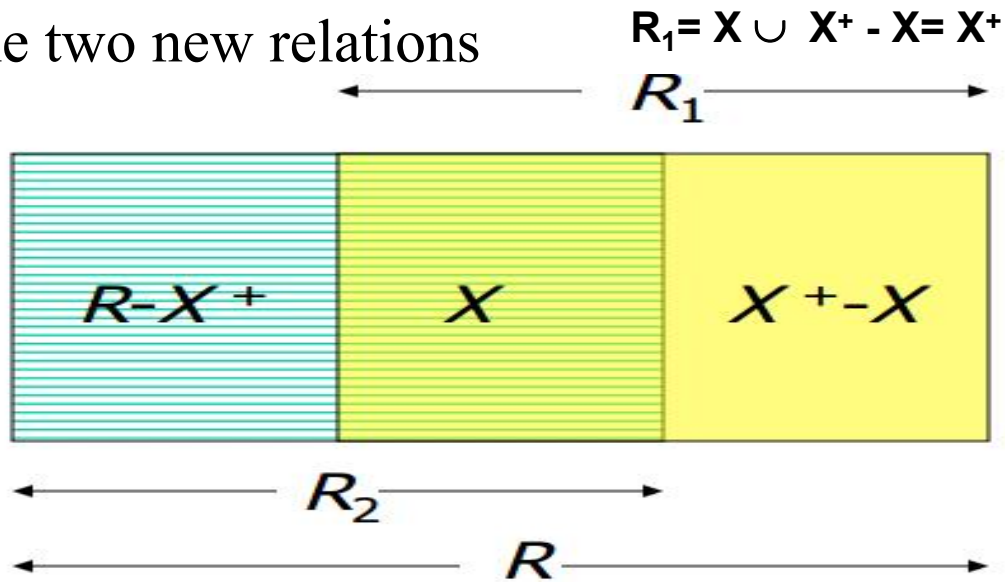
Not all attributes proceed to decomposition , or else X is a super key (so it will be satisfying BCNF)

Decompose R Using $X \rightarrow Y$

Replace R by relations with schemas:

1. $R_1 = X^+$ (note X^+ also includes Y)
2. $R_2 = R - (X^+ - X)$

Project given FD's F onto the two new relations



Example of BCNF Decomposition

Suppose we have a schema R and a non-trivial dependency $\alpha \rightarrow \beta$ causes a violation of BCNF.

We decompose R into:

- $(\alpha \cup \beta)$
- $(R - (\beta - \alpha))$

- ***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 (Cont.)

- BCNF Decomposition:
  - ***course_id*** $\rightarrow$ ***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 (Cont.)

- *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

- $R = (J, K, L)$

  $F = \{ JK \rightarrow L , \ L \rightarrow K \}$

  Two candidate keys = *JK* and *JL* *(see the similar example in slide)*

- ***R*** is **not in BCNF**
- As $L \rightarrow K$ ***fails*** , *we have to decompose into* $R_1(J,L)$ & $R_2(L,K)$
- Any decomposition of *R* will fail to preserve

$$JK \rightarrow L$$

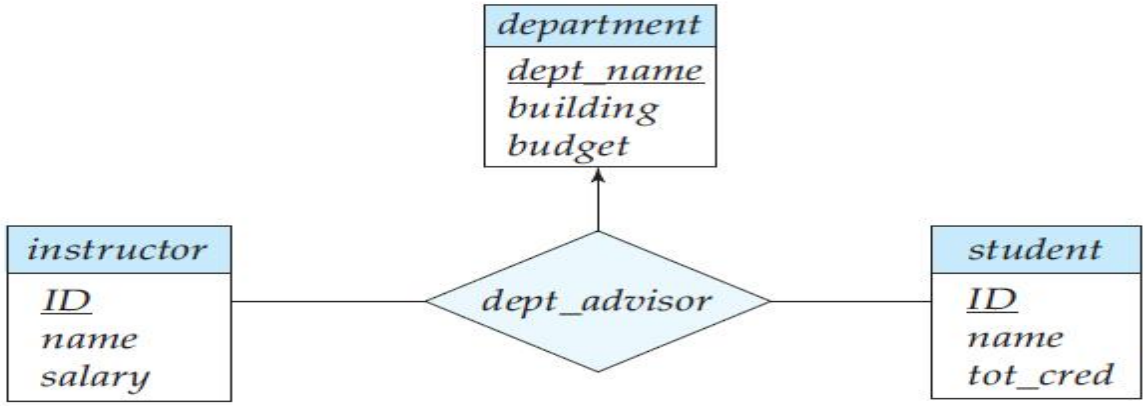
This implies that testing for $JK \rightarrow L$ requires a join

This example illustrate that when R is decomposed in to R_1 & R_2 satisfying BCNF, always it may not be Dependency Preserving.

"BCNF decomposition is not Always Dependency Preserving"

This is the Motivation for 3NF

This example also illustrates BCNF is Not always Dependency Preserving



The schema derived from *the ER diagram* is

Instructor(ID, name, salary) ***Student(ID, name, tot_cred)***

Dept_advisor (s ID, i ID, deptname)

As per the constraints the following functional dependencies hold on ***dept_advisor***:

i_ID→dept name

S_ID, deptname→i ID

Dept_advisor

S_Id	i_Id	DeptName
S1	I1	D1
S1	I2	D2
S2	I1	D1
S2	I2	D2
S3	I3	D1
NULL	I4	D1

Note: For a Instructor **I4** belonging to DeptName **D1** mqy not be advisor for any of the student ,so **S_id is NULL** for it.

- Note this is not BCNF
- As per decomposition rule we get following schema(for dept_advisor)-
  - **(*s_ID*, *i ID*)**
  - **(*i_ID*, *dept name*)**
- In this schema is BCNF but,
- we are unable to represent ***(S_ID, dept name) →i_ID FD.***
- ***Therefore FD is not preserved in BCNF.***

3 NF

A relation schema R is in **third normal form** 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**.
- α is a **super key** for R.
- Each attribute A **in** $\beta - \alpha$ is contained in a candidate key for R.

Note that the third condition above does not say that a single candidate key must contain all the attributes in $\beta - \alpha$; each attribute A in $\beta - \alpha$ may be **contained in a different candidate key.**

Any **"schema that satisfies BCNF also satisfies 3NF"**, since each of its functional dependencies would satisfy one of the first two alternatives. BCNF is therefore a more restrictive normal form than is 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 super key.**
- If α is **not a super key**, we have to **verify if each attribute in $\beta - \alpha$ is contained in a candidate key of *R***
  - Attributes in $\beta - \alpha$ may be in different candidate keys of R

3NF Example

- Relation ***dept_advisor***: (refer slide for dept_advisor example)
  - ***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* *not trivial*
      - ***(s_ID dept_name)*** **is a superkey 2nd condition satisfied**
    - *i_ID* $\rightarrow$ *dept_name* ; ***i_ID is not super key*** , *but*
      - *dept_name* is contained in a candidate key (s_ID dept_name) **3rd condition satisfied**

Test for 3NF-example

$R = (J, K, L)$

$F = \{JK \rightarrow L, L \rightarrow K\}$ **is R in 3NF ?**

- Take $JK \rightarrow L$, *is JK* is super key ?

  $JK^+ = \{ JKL\}$, yes, JK is super key, **1st condition for 3NF satisfied**

  Note JK is candidate key also (check yourself)

- Take $L \rightarrow K$ *is L* is super key ?

  $L^+ = \{ L\}$, no, L is not super key, **1st condition for 3NF NOT satisfied**

  K-L =K & K contained in candidate key (JK), **2nd condition for 3NF satisfied.**

"Therefore *R (J, K, L) is in 3NF*"

If you Replace J=s_ID ; K=dept_name & L= i_ID , this example is same as given in slide

Redundancy in 3NF

- There is some redundancy in this schema

  (the relation dept_advisor given in slide can also be discussed as below)

- **Example for problems due to redundancy in 3NF**
  - $R = (J, K, L)$
    $F = \{JK \rightarrow L, L \rightarrow K\}$

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

If you Replace J=s_ID ; K=dept_name & L= i_ID , this example is same as given in slide 75

- repetition of information (e.g., the relationship l_1, k_1)
  - i.e. (*i_ID, dept_name)* *in slide 75*
- need to use null values (e.g., to represent the relationship l_2, k_2 where there is no corresponding value for J).

(i.e. for an instructor in a department may not be an advisor for any student ,S_Id (i.e. J here is Null)

  - (*i_ID, dept_nameI)* if there is no separate relation mapping instructors to departments

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**

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 */

$R_j = R_i$;

$i=i-1$;

return $(R_1, R_2, ..., R_i)$

Note: algorithm ensures:

- each relation schema R_i is in 3NF
- decomposition is dependency preserving and lossless-join

3NF Decomposition example

Assume $R = (A, B, C)$

$F_c = \{A \rightarrow B\ ,\ B \rightarrow C\}$, **A is candidate key (if not given find yourself).**

***R* is not in 3 NF and Decompose to 3 NF**

Solution:

i=0

For loop

take $A \rightarrow B$, i=1 , R_1=A,B

take $B \rightarrow C$, i=2 , R_2 =B,C

IF none of R_1 ,R_2 contains Candidate key (i.e. A) is FALSE

because A is contained in R_1

Repeat

there is neither R_1 contained in R_2 nor R_2 contained in R_1

therefore No Deletion of duplicate schema

Return (R_1,R_2)

"R_1(A,B) & R_2(B,C) are decomposed 3 NF Relations"

3NF Decomposition: An Example

- Relation schema:

  cust_banker_branch = (*customer_id, employee_id, branch_name, type*)

  *Take customer_id, employee_id is the candidate key**

- The functional dependencies for this relation schema are:
  1. *customer_id, employee_id* → *branch_name, type*
  2. *employee_id* → *branch_name*
  3. *customer_id, branch_name* → *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* → *type*
employee_id → *branch_name*
customer_id, branch_name → *employee_id* }

* *Finding candidate key* *refer slide*

3NF Decompsition Example (Cont.)

- 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 namely- (*customer_id, branch_name, employee_id)*

  - 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)*

R(A,B,C,D,E,F,G,H) and F={ ABC →DE , E →BCG, F →AH }

Is it in 3NF Decompose into 3NF.

- Assume candidate keys are FE and FBC
- F_c ={ABC →DE , E →BCG, F →AH }
-

R(A,B,C,D,E,F,G,H,I) and F={ AB → C ,A → DE , B →F, F →GH, D →IJ}

Is it in 3NF Decompose into 3NF.

- Assume candidate keys are AB
- F_c={ AB → C ,A → DE , B →F, F →GH, D →IJ}

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.

Design Goals

- Goal for a relational database design is:
  - BCNF.
  - Lossless join.
  - Dependency preservation.
- If we cannot achieve this, we accept one of
  - Lack of dependency preservation
  - Redundancy due to use of 3NF
- Interestingly, SQL does not provide a direct way of specifying functional dependencies other than superkeys.

  Can specify FDs using assertions, but they are expensive to test, (and currently not supported by any of the widely used databases!)

Example

X (A, B, D, E, F, G, H, J) with A and B as primary key
Y (K, L, M, N, O, P, Q, R) with K and L as primary key

Let

F={A $\rightarrow$ G, H ; B $\rightarrow$ D, E, F; E $\rightarrow$ F; H $\rightarrow$ G; L $\rightarrow$ N, O, P, Q; N $\rightarrow$ Q

END