# Database Design and Normal Forms

Database Design

- coming up with a "good" schema is very important

How do we characterize the "goodness" of a schema ?
If two or more alternative schemas are available
how do we compare them ?
What are the problems with "bad" schema designs ?

Normal Forms:
Each normal form specifies certain conditions
If the conditions are satisfied by the schema
certain kind of problems are avoided

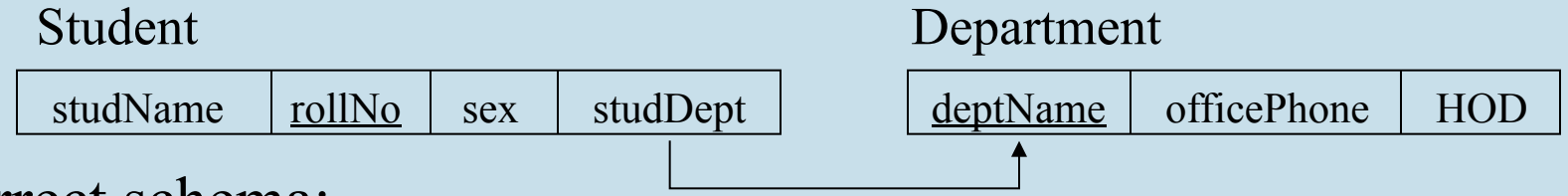
Details follow….

# An Example

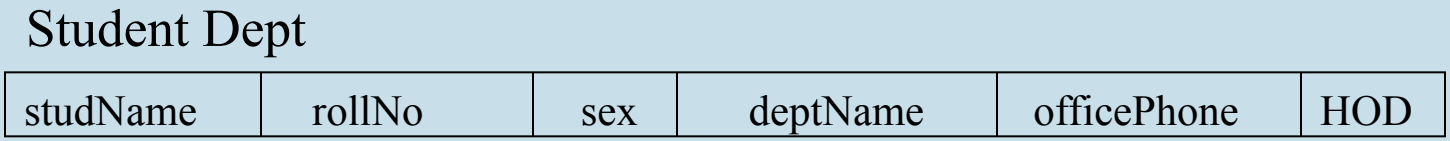
*student* relation with attributes: studName, rollNo, sex, studDept
*department* relation with attributes: deptName, officePhone, hod

Several students belong to a department.
studDept gives the name of the student's department.

Correct schema:




Incorrect schema:




What are the problems that arise ?

# Problems with bad schema

Redundant storage of data:

Office Phone & HOD info - stored redundantly

- once with each student that belongs to the department
- wastage of disk space

A program that updates Office Phone of a department

- must change it at several places
  - more running time
  - error - prone

Transactions running on a database

- must take as short time as possible to increase transaction throughput

# Update Anomalies

Another kind of problem with bad schema
Insertion anomaly:
  No way of inserting info about a new department unless
    we also enter details of a (dummy) student in the department

Deletion anomaly:
  If all students of a certain department leave
  and we delete their tuples,
  information about the department itself is lost

Update Anomaly:
  Updating officePhone of a department
  - value in several tuples needs to be changed
  - if a tuple is missed - inconsistency in data

# Normal Forms

First Normal Form (1NF) - included in the definition of a relation

Second Normal Form (2NF)

Third Normal Form (3NF)

Boyce-Codd Normal Form (BCNF)

} defined in terms of functional dependencies

Fourth Normal Form (4NF) - defined using multivalued dependencies

Fifth Normal Form (5NF) or Project Join Normal Form (PJNF) defined using join dependencies

# Functional Dependencies

A functional dependency (FD) $X \rightarrow Y$
(read as X *determines* Y) ($X \subseteq R$, $Y \subseteq R$)
is said to hold on a schema R if
in any instance r on R,
if two tuples $t_1$, $t_2$ ($t_1 \neq t_2$, $t_1 \in r$, $t_2 \in r$)
agree on X i.e. $t_1[X] = t_2[X]$
then they also agree on Y i.e. $t_1[Y] = t_2[Y]$

Note: If $K \subset R$ is a key for R then for any $A \in R$,
$K \rightarrow A$
holds because the above if …..then condition is vacuously true

# Functional Dependencies – Examples

Consider the schema:

Student ( studName, rollNo, sex, dept, hostelName, roomNo)

Since rollNo is a key, rollNo → {studName, sex, dept, hostelName, roomNo}

Suppose that each student is given a hostel room exclusively, then

hostelName, roomNo → rollNo

Suppose boys and girls are accommodated in separate hostels, then

hostelName → sex

FDs are additional constraints that can be specified by designers

# Trivial / Non-Trivial FDs

An FD $X \rightarrow Y$ where $Y \subseteq X$

- called a *trivial* FD, it always holds good

An FD $X \rightarrow Y$ where $Y \not\subseteq X$

- *non-trivial* FD

An FD $X \rightarrow Y$ where $X \cap Y = \Phi$

- *completely non-trivial* FD

# Deriving new FDs

Given that a set of FDs F holds on R

we can infer that a certain new FD must also hold on R

For instance,

given that $X \rightarrow Y$, $Y \rightarrow Z$ hold on R

we can infer that $X \rightarrow Z$ must also hold

How to systematically obtain all such new FDs ?

Unless *all* FDs are known, a relation schema is not fully specified

# Entailment relation

We say that a set of FDs $F \models \{ X \rightarrow Y \}$

(read as F *entails* $X \rightarrow Y$ or

F *logically implies* $X \rightarrow Y$)

if in every instance r of R on which FDs F hold,

FD $X \rightarrow Y$ also holds.

Armstrong came up with several inference rules

for deriving new FDs from a given set of FDs

We define $F^+ = \{X \rightarrow Y \mid F \models X \rightarrow Y\}$

$F^+$: Closure of F

# Armstrong's Inference Rules (1/2)

1. Reflexive rule

   $F \models \{X \rightarrow Y \mid Y \subseteq X\}$ for any X. Trivial FDs

2. Augmentation rule

   $\{X \rightarrow Y\} \models \{XZ \rightarrow YZ\}, Z \subseteq R.$ Here XZ denotes $X \cup Z$

3. Transitive rule

   $\{X \rightarrow Y, Y \rightarrow Z\} \models \{X \rightarrow Z\}$

4. Decomposition or Projective rule

   $\{X \rightarrow YZ\} \models \{X \rightarrow Y\}$

5. Union or Additive rule

   $\{X \rightarrow Y, X \rightarrow Z\} \models \{X \rightarrow YZ\}$

6. Pseudo transitive rule

   $\{X \rightarrow Y, WY \rightarrow Z\} \models \{WX \rightarrow Z\}$

# Armstrong's Inference Rules (2/2)

Rules 4, 5, 6 are not really necessary.

For instance, Rule 5: $\{X \rightarrow Y, X \rightarrow Z\} \models \{X \rightarrow YZ\}$ can be proved using 1, 2, 3 alone

1) $X \rightarrow Y$
2) $X \rightarrow Z$

} given

3) $X \rightarrow XY$ Augmentation rule on 1
4) $XY \rightarrow ZY$ Augmentation rule on 2
5) $X \rightarrow ZY$ Transitive rule on 3, 4.

Similarly, 4, 6 can be shown to be unnecessary.
But it is useful to have 4, 5, 6 as short-cut rules

# Sound and Complete Inference Rules

Armstrong showed that

Rules (1), (2) and (3) are sound and complete.

These are called Armstrong's Axioms (AA)

Soundness:

Every new FD $X \rightarrow Y$ derived from a given set of FDs F using Armstrong's Axioms is such that $F \models \{X \rightarrow Y\}$

Completeness:

Any FD $X \rightarrow Y$ logically implied by F (i.e. $F \models \{X \rightarrow Y\}$) can be derived from F using Armstrong's Axioms

# Proving Soundness

Suppose $X \rightarrow Y$ is derived from F using AA in some *n* steps.

If each step is correct then overall deduction would be correct.

Single step: Apply Rule (1) or (2) or (3)

Rule (1) – obviously results in correct FDs

Rule (2) – $\{X \rightarrow Y\} \models \{XZ \rightarrow YZ\}, Z \subseteq R$

Suppose $t_1, t_2 \in$ r agree on XZ

$\Rightarrow t_1, t_2$ agree on X

$\Rightarrow t_1, t_2$ agree on Y (since $X \rightarrow Y$ holds on r)

$\Rightarrow t_1, t_2$ agree as YZ

Hence Rule (2) gives rise to correct FDs

Rule (3) – $\{X \rightarrow Y, Y \rightarrow Z\} \models X \rightarrow Z$

Suppose $t_1, t_2 \in$ r agree on X

$\Rightarrow t_1, t_2$ agree on Y (since $X \rightarrow Y$ holds)

$\Rightarrow t_1, t_2$ agree on Z (since $Y \rightarrow Z$ holds)

# Proving Completeness of Armstrong's Axioms (1/4)

Define $X^+_F$ (closure of X wrt F)

$= \{A \mid X \rightarrow A$ can be derived from F using AA$\}$, $A \in R$

Claim1:

$X \rightarrow Y$ can be derived from F using AA iff $Y \subseteq X^+$

(If) Let $Y = \{A_1, A_2, \ldots, A_n\}$. $Y \subseteq X^+$

$\Rightarrow X \rightarrow A_i$ can be derived from F using AA $(1 \leq i \leq n)$

By union rule, it follows that $X \rightarrow Y$ can be derived from F.

(Only If) $X \rightarrow Y$ can be derived from F using AA

By projective rule $X \rightarrow A_i$ $(1 \leq i \leq n)$

Thus by definition of $X^+$, $A_i \in X^+$

$\Rightarrow Y \subseteq X^+$

# Completeness of Armstrong's Axioms (2/4)

Completeness:

$(F \models \{X \rightarrow Y\}) \Rightarrow X \rightarrow Y$ follows from F using AA

We will prove the contrapositive:

$X \rightarrow Y$ can't be derived from F using AA

$\Rightarrow F \not\models \{X \rightarrow Y\}$

$\Rightarrow \exists$ a relation instance r on R st all the FDs of F hold on r but $X \rightarrow Y$ doesn't hold.

Consider the relation instance r with just two tuples:

| | $X^+$ attributes | | | Other attributes | | | |
|---|---|---|---|---|---|---|---|
| r: | 1 | 1 | 1 …1 | 1 | 1 | 1 | …1 |
| | 1 | 1 | 1 …1 | 0 | 0 | 0 | …0 |

# Completeness Proof (3/4)

Claim 2: All FDs of F are satisfied by r

Suppose not. Let $W \rightarrow Z$ in F be an FD not satisfied by r

Then $W \subseteq X^+$ and $Z \notin X^+$

Let $A \in Z - X^+$

Now, $X \rightarrow W$ follows from F using AA as $W \subseteq X^+$ (claim 1)

$X \rightarrow Z$ follows from F using AA by transitive rule

$Z \rightarrow A$ follows from F using AA by reflexive rule as $A \in Z$

$X \rightarrow A$ follows from F using AA by transitive rule

By definition of closures, A must belong to $X^+$

- a contradiction.

Hence the claim.

r:

| 1 | 1 | 1 | …1 | 1 | 1 | 1 | …1 |
|---|---|---|---|---|---|---|---|
| 1 | 1 | 1 | …1 | 0 | 0 | 0 | …0 |

$X^+$ $\quad$ $R - X^+$

# Completeness Proof (4/4)

Claim 3: $X \rightarrow Y$ is not satisfied by r

Suppose not

Because of the structure of r, $Y \subseteq X^+$

$\Rightarrow X \rightarrow Y$ can be derived from F using AA

contradicting the assumption about $X \rightarrow Y$

Hence the claim

Thus, whenever $X \rightarrow Y$ doesn't follow from F using AA,

F doesn't logically imply $X \rightarrow Y$

Armstrong's Axioms are complete.

# Consequence of Completeness of AA

$$X^+ = \{A \mid X \rightarrow A \text{ follows from F using AA}\}$$

$$= \{A \mid F \models X \rightarrow A\}$$

Similarly

$$F^+ = \{X \rightarrow Y \mid F \models X \rightarrow Y\}$$

$$= \{X \rightarrow Y \mid X \rightarrow Y \text{ follows from F using AA}\}$$

# Computing closures

The size of $F^+$ can sometimes be exponential in the size of F.

For instance, $F = \{A \rightarrow B_1, A \rightarrow B_2, \ldots.., A \rightarrow B_n\}$

$F^+ = \{A \rightarrow X\}$ where $X \subseteq \{B_1, B_2, \ldots, B_n\}$.

Thus $|F^+| = 2^n$

Computing $F^+$: computationally expensive

Fortunately, checking if $X \rightarrow Y \in F^+$

can be done by checking if $Y \subseteq X^+_F$

Computing attribute closure ($X^+_F$) is easier

# Computing $X^+_F$

We compute a sequence of sets $X_0, X_1, \ldots$ as follows:

$X_0 := X$; // X is the given set of attributes

$X_{i+1} := X_i \cup \{A \mid$ there is a FD $Y \rightarrow Z$ in F and $A \in Z$ and $Y \subseteq X_i\}$

Since $X_0 \subseteq X_1 \subseteq X_2 \subseteq \ldots \subseteq X_i \subseteq X_{i+1} \subseteq \ldots \subseteq R$
and R is finite,
There is an integer i st $X_i = X_{i+1} = X_{i+2} = \ldots$
and $X^+_F$ is equal to $X_i$.

# Normal Forms – 2NF

Full functional dependency:

An FD $X \rightarrow A$ for which there is no proper subset Y of X such that $Y \rightarrow A$

(A is said to be fully functionally dependent on X)

2NF: A relation schema R is in 2NF if

every non-prime attribute is fully functionally dependent on any key of R

prime attribute: A attribute that is part of some key

non-prime attribute: An attribute that is not part of any key

# Example

1) Book (authorName, title, authorAffiliation, ISBN, publisher, pubYear )

Keys: (authorName, title), ISBN

Not in 2NF as authorName → authorAffiliation

(authorAffiliation is not fully functionally dependent on the first key)

2) Student (rollNo, name, dept, sex, hostelName, roomNo, admitYear)

Keys: rollNo, (hostelName, roomNo)

Not in 2NF as hostelName → sex

student (rollNo, name, dept, hostelName, roomNo, admitYear)

hostelDetail (hostelName, sex)

- There are both in 2NF

# Transitive Dependencies

Transitive dependency:

An FD $X \rightarrow Y$ in a relation schema R for which there is a set of attributes $Z \subseteq R$ such that

$X \rightarrow Z$ and $Z \rightarrow Y$ and Z is not a subset of any key of R

student (rollNo, name, dept, hostelName, roomNo, headDept)

Keys: rollNo, (hostelName, roomNo)

rollNo → dept; dept → headDept hold

So, rollNo → headDept a transitive dependency

Head of the dept of dept D is stored redundantly in every tuple where D appears.

Relation is in 2NF but redundancy still exists.

# Normal Forms – 3NF

Relation schema R is in 3NF if it is in 2NF and no non-prime attribute of R is transitively dependent on any key of R

student (rollNo, name, dept, hostelname, roomNo, headDept)
is not in 3NF

Decompose: student (rollNo, name, dept, hostelName, roomNo)
deptInfo (dept, headDept)
both in 3NF

Redundancy in data storage - removed

# Another definition of 3NF

Relation schema R is in 3NF if for any nontrivial FD $X \rightarrow A$
either (i) X is a superkey or (ii) A is prime.

Suppose some R violates the above definition
$\Rightarrow$ There is an FD $X \rightarrow A$ for which both (i) and (ii) are false
$\Rightarrow$ X is not a superkey and A is non-prime attribute

Two cases arise:
1) X is contained in a key – A is not fully functionally dependent on this key
   - violation of 2NF condition
2) X is not contained in a key
   $K \rightarrow X$, $X \rightarrow A$ is a case of transitive dependency
   (K – any key of R)

# Motivating example for BCNF

gradeInfo (rollNo, studName, course, grade)

Suppose the following FDs hold:

1) rollNo, course → grade
2) studName, course → grade
3) rollNo → studName
4) studName → rollNo

Keys:

(rollNo, course)
(studName, course)

For 1,2 lhs is a key. For 3,4 rhs is prime
So gradeInfo is in 3NF

But studName is stored redundantly along with every course being done by the student

# Boyce - Codd Normal Form (BCNF)

Relation schema R is in BCNF if for every nontrivial
FD $X \rightarrow A$, X is a *superkey* of R.

In gradeInfo, FDs 3, 4 are nontrivial but lhs is not a superkey
So, gradeInfo is not in BCNF

Decompose:

gradeInfo (rollNo, course, grade)
studInfo (rollNo, studName)

Redundancy allowed by 3NF is disallowed by BCNF

BCNF is stricter than 3NF
3NF is stricter than 2NF

# Decomposition of a relation schema

If R doesn't satisfy a particular normal form,
we decompose R into smaller schemas

What's a decomposition?

$R = (A_1, A_2, \ldots, A_n)$

$D = (R_1, R_2, \ldots, R_k)$ st $R_i \subseteq R$ and $R = R_1 \cup R_2 \cup \ldots \cup R_k$

($R_i$'s need not be disjoint)

Replacing R by $R_1, R_2, \ldots, R_k$ is the process of decomposing R

Ex: gradeInfo (rollNo, studName, course, grade)

$R_1$: gradeInfo (<u>rollNo, course</u>, grade)

$R_2$: studInfo (<u>rollNo</u>, studName)

# Desirable Properties of Decompositions

Not all decomposition of a schema are useful

We require two properties to be satisfied

(i) Lossless join property

- the information in an instance r of R must be preserved in the instances $r_1, r_2, \ldots, r_k$ where $r_i = \Pi_{R_i}(r)$

(ii) Dependency preserving property

- if a set F of dependencies hold on R it should be possible to enforce F by enforcing appropriate dependencies on each $r_i$

# Lossless join property

F – set of FDs that hold on R

R – decomposed into $R_1, R_2, \ldots, R_k$

Decomposition is *lossless* wrt F if

for every relation instance r on R satisfying F,

$r = \Pi_{R_1}(r) * \Pi_{R_2}(r) * \ldots * \Pi_{R_k}(r)$

Lossless joins are also called non-additive joins

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

Original info is distorted

r:

| A | B | C |
|---|---|---|
| $a_1$ | $b_1$ | $c_1$ |
| $a_2$ | $b_2$ | $c_2$ |
| $a_3$ | $b_1$ | $c_3$ |

$r_1$:

| A | B |
|---|---|
| $a_1$ | $b_1$ |
| $a_2$ | $b_2$ |
| $a_3$ | $b_1$ |

$r_2$:

| B | C |
|---|---|
| $b_1$ | $c_1$ |
| $b_2$ | $c_2$ |
| $b_1$ | $c_3$ |

$r_1 * r_2$:

| A | B | C | |
|---|---|---|---|
| $a_1$ | $b_1$ | $c_1$ | |
| $a_1$ | $b_1$ | $c_3$ | Spurious tuples |
| $a_2$ | $b_2$ | $c_2$ | |
| $a_3$ | $b_1$ | $c_1$ | Spurious tuples |
| $a_3$ | $b_1$ | $c_3$ | |

Lossy join

# Dependency Preserving Decompositions

Decomposition $D = (R_1, R_2, \ldots, R_k)$ of schema R *preserves* a set of dependencies F if

$$(\Pi_{R_1}(F) \cup \Pi_{R_2}(F) \cup \ldots \cup \Pi_{R_k}(F))^+ = F^+$$

Here, $\Pi_{R_i}(F) = \{ (X \rightarrow Y) \in F^+ \mid X \subseteq R_i, Y \subseteq R_i \}$
(called projection of F onto $R_i$)

Informally, any FD that logically follows from F must also logically follow from the union of projections of F onto $R_i$'s
Then, D is called dependency preserving.

# An example

Schema R = (A, B, C)
FDs F = $\{A \rightarrow B, B \rightarrow C, C \rightarrow A\}$

Decomposition D = ($R_1 = \{A, B\}$, $R_2 = \{B, C\}$)
$\Pi_{R_1}(F) = \{A \rightarrow B, B \rightarrow A\}$
$\Pi_{R_2}(F) = \{B \rightarrow C, C \rightarrow B\}$

$(\Pi_{R_1}(F) \cup \Pi_{R_2}(F))^+ = \{A \rightarrow B, B \rightarrow A,$
$B \rightarrow C, C \rightarrow B,$
$A \rightarrow C, C \rightarrow A\} = F^+$

Hence Dependency preserving

# Testing for lossless decomposition property(1/6)

R – given schema with attributes $A_1, A_2, \ldots, A_n$
F – given set of FDs
D – $\{R_1, R_2, \ldots, R_m\}$ given decomposition of R

Is D a lossless decomposition?

Create an $m \times n$ matrix $S$ with columns labeled as $A_1, A_2, \ldots, A_n$
and rows labeled as $R_1, R_2, \ldots, R_m$

Initialize the matrix as follows:
set $S(i,j)$ as symbol $b_{ij}$ for all $i,j$.
if $A_j$ is in the scheme $R_i$, then set $S(i,j)$ as symbol $a_j$ , for all $i,j$

# Testing for lossless decomposition property(2/6)

After *S* is initialized, we carry out the following process on it:

**repeat**
  **for each** functional dependency $U \rightarrow V$ in *F* **do**
    **for all** rows in *S* which agree on *U*-attributes **do**
      make the symbols in each *V*- attribute column
        the *same* in all the rows as follows:
          if any of the rows has an "*a*" symbol for the column
            set the other rows to the same "*a*" symbol in the column
          else // if no "*a*" symbol exists in any of the rows
            choose one of the "*b*" symbols that appears
            in one of the rows for the *V*-attribute and
            set the other rows to that "b" symbol in the column
**until** no changes to S

At the end, if there exists a row with all "*a*" symbols then D is lossless otherwise D is a lossy decomposition

# Testing for lossless decomposition property(3/6)

R = (rollNo, name, advisor, advisorDept, course, grade)

FD's = { rollNo → name; rollNo → advisor; advisor → advisorDept
rollNo, course → grade}

D : { $R_1$ = (rollNo, name, advisor), $R_2$ = (advisor, advisorDept),
$R_3$ = (rollNo, course, grade) }

Matrix S : (Initial values)

| | rollNo | name | advisor | advisor Dept | course | grade |
|---|---|---|---|---|---|---|
| $R_1$ | $a_1$ | $a_2$ | $a_3$ | $b_{14}$ | $b_{15}$ | $b_{16}$ |
| $R_2$ | $b_{21}$ | $b_{22}$ | $a_3$ | $a_4$ | $b_{25}$ | $b_{26}$ |
| $R_3$ | $a_1$ | $b_{32}$ | $b_{33}$ | $b_{34}$ | $a_5$ | $a_6$ |

# Testing for lossless decomposition property(4/6)

R = (rollNo, name, advisor, advisorDept, course, grade)

FD's = { rollNo → name; rollNo → advisor; advisor → advisorDept
rollNo, course → grade}

D : { $R_1$ = (rollNo, name, advisor), $R_2$ = (advisor, advisorDept),
$R_3$ = (rollNo, course, grade) }

Matrix S : (After enforcing rollNo → name & rollNo → advisor)

| | rollNo | name | advisor | advisor Dept | course | grade |
|---|---|---|---|---|---|---|
| $R_1$ | $a_1$ | $a_2$ | $a_3$ | $b_{14}$ | $b_{15}$ | $b_{16}$ |
| $R_2$ | $b_{21}$ | $b_{22}$ | $a_3$ | $a_4$ | $b_{25}$ | $b_{26}$ |
| $R_3$ | $a_1$ | ~~$b_{32}$~~ $a_2$ | ~~$b_{33}$~~ $a_3$ | $b_{34}$ | $a_5$ | $a_6$ |

# Testing for lossless decomposition property(5/6)

R = (rollNo, name, advisor, advisorDept, course, grade)

FD's = { rollNo → name; rollNo → advisor; advisor → advisorDept

rollNo, course → grade}

D : { $R_1$ = (rollNo, name, advisor), $R_2$ = (advisor, advisorDept),

$R_3$ = (rollNo, course, grade) }

Matrix S : (After enforcing advisor → advisorDept )

| | rollNo | name | advisor | advisor Dept | course | grade |
|---|---|---|---|---|---|---|
| $R_1$ | $a_1$ | $a_2$ | $a_3$ | ~~$b_{14}$~~ $a_4$ | $b_{15}$ | $b_{16}$ |
| $R_2$ | $b_{21}$ | $b_{22}$ | $a_3$ | $a_4$ | $b_{25}$ | $b_{26}$ |
| $R_3$ | $a_1$ | ~~$b_{32}$~~ $a_2$ | ~~$b_{33}$~~ $a_3$ | ~~$b_{34}$~~ $a_4$ | $a_5$ | $a_6$ |

No more changes. Third row with all *a* symbols. So a lossless join.

# Testing for lossless decomposition property(6/6)

R – given schema. F – given set of FDs

The decomposition of R into $R_1$, $R_2$ is lossless wrt F if and only if

either $R_1 \cap R_2 \rightarrow (R_1 - R_2)$ belongs to $F^+$ or

$R_1 \cap R_2 \rightarrow (R_2 - R_1)$ belongs to $F^+$

Eg. gradeInfo (rollNo, studName, course, grade)

with FDs = {rollNo, course → grade; studName, course → grade;
rollNo → studName; studName → rollNo}

decomposed into

grades (rollNo, course, grade) and studInfo (rollNo, studName)

is lossless because

rollNo → studName

# A property of lossless joins

$D_1$: $(R_1, R_2, \ldots, R_K)$ lossless decomposition of R wrt F

$D_2$: $(R_{i1}, R_{i2}, \ldots, R_{ip})$ lossless decomposition of $R_i$ wrt $F_i = \Pi_{R_i}(F)$

Then

$D = (R_1, R_2, \ldots, R_{i-1}, R_{i1}, R_{i2}, \ldots, R_{ip}, R_{i+1}, \ldots, R_k)$ is a lossless decomposition of R wrt F

This property is useful in the algorithm for BCNF decomposition

# Algorithm for BCNF decomposition

R – given schema.  F – given set of FDs

D = {R}  // initial decomposition
while there is a relation schema $R_i$ in D that is not in BCNF do
{ let $X \rightarrow A$ be the FD in $R_i$ violating BCNF;
  Replace $R_i$ by $R_{i1} = R_i - \{A\}$ and $R_{i2} = X \cup \{A\}$ in D;
}

Decomposition of $R_i$ is lossless as

$$R_{i1} \cap R_{i2} = X,\ R_{i2} - R_{i1} = A \text{ and } X \rightarrow A$$

Result: a lossless decomposition of R into BCNF relations

# Dependencies may not be preserved (1/2)

S T D

Consider the schema: townInfo (stateName, townName, distName)
with the FDs F: ST → D (town names are unique within a state)
D → S (district names are unique across states)

Keys: ST, DT. – all attributes are prime
– relation in 3NF

Relation is not in BCNF as D → S and D is not a key

Decomposition given by algorithm: R1: TD R2: DS

Not dependency preserving as $\Pi_{R1}(F)$ = trivial dependencies
$\Pi_{R2}(F) = \{D \rightarrow S\}$

Union of these doesn't imply ST → D

ST → D can't be enforced unless we perform a join.

# Dependencies may not be preserved (2/2)

Consider the schema: R (A, B, C)
with the FDs F: AB → C and C → B
Keys: AB, AC – relation in 3NF (all attributes are prime)
– Relation is not in BCNF as C → B and C is not a key

Decomposition given by algorithm: $R_1$: CB $R_2$: AC
Not dependency preserving as $\Pi_{R_1}(F)$ = trivial dependencies
$\Pi_{R_2}(F) = \{C \rightarrow B\}$
Union of these doesn't imply AB → C

All possible decompositions: {AB, BC}, {BA, AC}, {AC, CB}
Only the last one is lossless!

Lossless and dependency-preserving decomposition doesn't exist.

# Equivalent Dependency Sets

F, G – two sets of FDs on schema R

F is said to cover G if $G \subseteq F^+$ (equivalently $G^+ \subseteq F^+$)

F is equivalent to G if $F^+ = G^+$ (or, F covers G and G covers F)

Note: To check if F covers G,

it's enough to show that for each FD $X \rightarrow Y$ in G, $Y \subseteq X^+_F$

# Canonical covers or Minimal covers

It is of interest to reduce a set of FDs F into a 'standard' form F′ such that F′ is equivalent to F.

We define that a set of FDs F is in '*minimal form*' if

(i) the rhs of any FD of F is a single attribute

(ii) there are no redundant FDs in F

that is, there is no FD $X \rightarrow A$ in F

s.t $(F - \{X \rightarrow A\})$ is equivalent to F

(iii) there are no redundant attributes on the lhs of any FD in F

that is, there is no FD $X \rightarrow A$ in F s.t there is $Z \subset X$ for which

$F - \{X \rightarrow A\} \cup \{Z \rightarrow A\}$ is equivalent to F

Minimal Covers

useful in obtaining a lossless, dependency-preserving decomposition of a scheme R into 3NF relation schemas

# Algorithm for computing a minimal cover

R – given Schema or set of attributes; F – given set of fd's on R

Step 1: G := F

Step 2: Replace every fd of the form $X \rightarrow A_1A_2A_3\ldots A_k$ in G
by $X \rightarrow A_1$; $X \rightarrow A_2$; $X \rightarrow A_3$; … ; $X \rightarrow A_k$

Step 3: For each fd $X \rightarrow A$ in G do
for each B in X do
if $A \in (X - B)^+$ wrt G then
replace $X \rightarrow A$ by $(X - B) \rightarrow A$

Step 4: For each fd $X \rightarrow A$ in G do
if $(G - \{ X \rightarrow A\})^+ = G^+$ then
replace G by $G - \{ X \rightarrow A\}$

# 3NF decomposition algorithm

R – given Schema; F – given set of fd's on R in *minimal form*

Use BCNF algorithm to get a lossless decomposition D = ($R_1$, $R_2$,…,$R_k$)
Note: each $R_i$ is already in 3NF (it is in BCNF in fact!)

Algorithm: Let G be the set of fd's not preserved in D
For each fd $Z \rightarrow A$ that is in G
Add relation scheme S = ($B_1$,$B_2$, …, $B_s$,A) to D. // Z = {$B_1$,$B_2$, …, $B_s$}

As $Z \rightarrow A$ is in F which is a minimal cover,
there is no proper subset X of Z s.t $X \rightarrow A$. So Z is a key for S!
Any other fd $X \rightarrow C$ on S is such that C is in {$B_1$,$B_2$, …, $B_s$}.
Such fd's do not violate 3NF because each $B_j$'s is prime a attribute!
Thus any scheme S added to D as above is in 3NF.

D continues to be lossless even when we add new schemas to it!

# Multi-valued Dependencies (MVDs) and 4NF

studCoursesAndFriends(rollNo,courseNo,frndEmailAddr)

A student enrolls for several courses and has several friends whose email addresses we want to record.

If rows (CS05B007, CS370, shyam@gmail.com) and
(CS05B007, CS376, radha@yahoo.com) appear then
rows (CS05B007, CS376, shyam@gmail.com)
(CS05B007, CS370, radha@yahoo.com) should also appear!

For, otherwise, it implies that having "shyam" as a friend has something to do with doing course CS370!

Causes a huge amount of data redundancy!
Since there are no non-trivial FD's, the scheme is in BCNF

We say that MVD rollNo $\rightarrow\!\!\rightarrow$ courseNo holds
(read as rollNo *multi-determines* courseNo)
By symmetry, rollNo $\rightarrow\!\!\rightarrow$ frndEmailAddr also holds

# More about MVDs

Consider studCourseGrade(rollNo,courseNo,grade)

Note that rollNo →→ courseNo *does not* hold here even though courseNo is a multi-valued attribute of a student entity

If (CS05B007, CS370, A)

(CS05B007, CS376, B) appear in the data then

(CS05B007, CS376, A)

(CS05B007, CS370, B) will not appear !!

Attribute 'grade' depends on (rollNo,courseNo)

MVD's arise when two or more *unrelated* multi-valued attributes of an entity are sought to be represented together in a scheme.

# More about MVDs

Consider

studCourseAdvisor(rollNo,courseNo,advisor)

Note that rollNo →→ courseNo *holds* here

If (CS05B007, CS370, Dr Ravi)

(CS05B007, CS376, Dr Ravi) appear in the data then

swapping courseNo values gives rise to existing rows only.

But, since rollNo → advisor and (rollNo, courseNo) is the key, this gets caught in checking for 2NF itself.

# MVD Definition

Consider a scheme R(X, Y, Z),

An MVD X →→Y holds on R if, for in any instance of R,

the presence of two tuples

(xxx, y1y1y1, z1z1z1) and

(xxx, y2y2y2, z2z2z2)

guarantees the presence of tuples

(xxx, y1y1y1, z2z2z2) and

(xxx, y2y2y2, z1z1z1)

Note that every FD on R is also an MVD!

- the notion of MVD's generalizes the notion of FD's

# Alternative definition of MVDs

Consider R(<u>X,Y,Z</u>)

Suppose that $X \rightarrow\rightarrow Y$ and by symmetry $X \rightarrow\rightarrow Z$

Then, decomposition D = (XY, XZ) of R should be lossless

That is, for any instance *r* on R, $r = \Pi_{XY}(r) * \Pi_{XZ}(r)$

# MVDs and 4NF

An MVD X →→ Y on scheme R is called *trivial* if either Y ⊆ X or R = X ∪ Y. Otherwise, it is called *non-trivial*.

**4NF**: A relation R is in 4NF if it is in BCNF and for every nontrivial MVD X →→ A, X must be a superkey of R.

studCourseEmail(rollNo,courseNo,frndEmailAddr)

is not in 4NF as

rollNo →→ courseNo and

rollNo →→ frndEmailAddr

are both nontrivial and rollNo is not a superkey for the relation

# Join Dependencies and 5NF

A join dependency (JD) is generalization of an MVD

A JD $JD(R_1,R_2,\ldots,R_k)$ is said to hold on r(R) if

for every instance $r = \Pi_{R1}(r) * \Pi_{R2}(r) * \ldots * \Pi_{Rk}(r)$

Here, $R = R_1 \cup R_2 \cup \ldots \cup R_k$

A JD is difficult to detect in practice. It occurs in rare situations.

A relational scheme is said to be in 5NF wrt to a set of FDs, MVDs and JDs if it is in 4NF and for every non-trivial $JD(R_1,R_2,\ldots,R_k)$, each $R_i$ is a superkey.

# Join Dependencies – An Example

Consider the following relation:

**offers**(rollNo, skill, project) and the three relations

**studSkill**(rollNo, skill) // who has what skill

**studProj**(rollNo, project) // who is interested in what project

**skillProj**(project, skill) // which project requires what skills

Suppose there is a rule that:

if a student d1 has skill s1, and d1 is interested in project p1

and project p1 requires skill s1 then (d1, s1, p1) *must be* in offers

In other words, offers = studSkill * studProj * skillProj

Then, we say JD(studSkill, studProj, skillProj) holds

# Join Dependencies – An Example

Suppose a student d2 has 4 different skills {s1, s2, s3, s4} and

d2 is interested in 4 different projects {p1, p2, p3, p4} and

all those 4 projects require all the 4 skills

Then.. to capture this info in the 3 relations,

4 + 4 + 16 = 24 tuples are needed

to capture the same info in offers, we need

16 + 16 = 32 tuples

Presence of JD's can cause data redundancy

Recommendation: decompose relation **offers** into the three relations