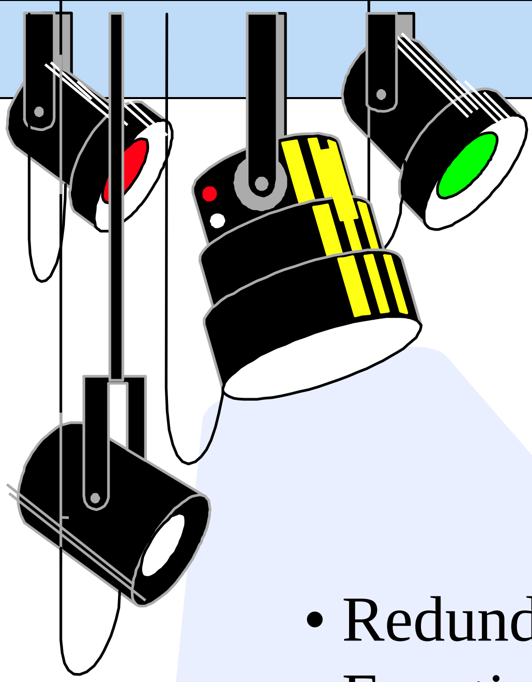


# Outline: Normalization

- Redundant information and update anomalies
- Function dependencies
- Normal forms
  - 1NF, 2NF, 3NF
  - BCNF (Boyce Codd normal form)
- Lossless join property

# Normalization

Reading:

14.1.2 Redundant … update anomalies

14.2.1 Functional dependencies

14.2.2 Inference rules for FDs

14.2.3 Equivalence of sets of FDs

14.2.4 Minimal sets of FDs

14.3 Normal forms based on PKs

# Normalization

Motivation:

Certain relation schemas have redundancy and update anomalies

- they may be difficult to understand and maintain

Normalization theory recognizes this and gives us some principles to guide our designs

Normal Forms: 1NF, 2NF, 3NF, BCNF, … are each an improvement on the previous ones in the list

Normalization is a process that generates higher normal forms.  Denormalization moves from higher to lower forms and might be applied for performance reasons.

Suppose we have the following relation

**EmployeeProject**

| **<u>ssn</u>** | **<u>pnumber</u>** | hours | ename | plocation |
|---|---|---|---|---|

This is similar to *Works_on*, but we have included *ename* and *plocation*

Suppose we have the following relation

**EmployeeDepartment**

| ename | <u>ssn</u> | bdate | address | dnumber | dname |
|---|---|---|---|---|---|

This is similar to *Employee*, but we have included *dname*

# Normalization

In the two prior cases with EmployeeDepartment and EmployeeProject, we have **redundant** information in the database …

- if two employees work in the same department, then that department name is replicated
- if more than one employee works on a project then the project location is replicated
- if an employee works on more than one project his/her name is replicated

Redundant data leads to

- additional space requirements
- update **anomalies**

Suppose EmployeeDepartment is the only relation where department name is recorded

insert **anomalies**

- adding a new department is complicated unless there is also an employee for that department

deletion **anomalies**

- if we delete all employees for some department, what should happen to the department information?

modification **anomalies**

- if we change the name of a department, then we must change it in all tuples referring to that department

# Normalization

If we design a database with a relation such as EmployeeDepartment then we will have complex update rules to enforce.

- difficult to code correctly
- will not be as efficient as possible

Such designs mix concepts.

For example, EmployeeDepartment mixes the Employee and Department concept

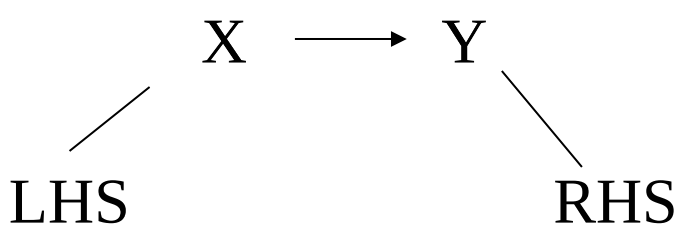
## Section 14.2 Functional dependencies

Suppose we have a relation R comprising attributes X,Y, …

We say a functional dependency exists between the attributes X and Y,

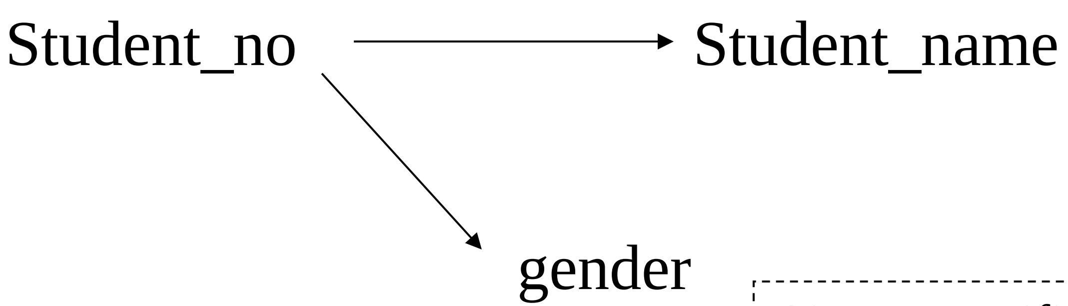
$$X \longrightarrow Y$$

if, whenever a tuple exists with the value x for X, it will always have the same value y for Y.



**Student**

| course_no | student_no | student_name | gender |
|---|---|---|---|




Given a specific student number, there is only one value for student name and only one value for gender found with it.

We always have functional dependencies between any candidate key and the other attributes.

**Student**

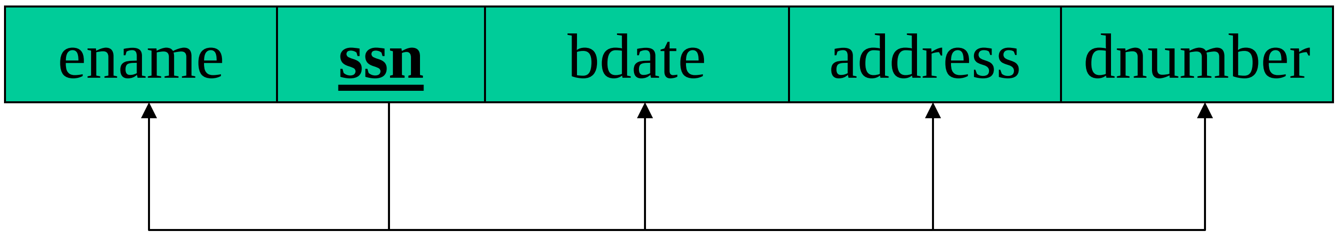
| <u>**student_no**</u> | student_name | student_address | gender |
|---|---|---|---|

*student_no* is unique … given a specific *student_no* there is only one *student name*, only one *student address*, only one *gender*

Student_no $\rightarrow$ student_name,
Student_no $\rightarrow$ student_address,
Student_no $\rightarrow$ gender

# Normalization

**Employee**



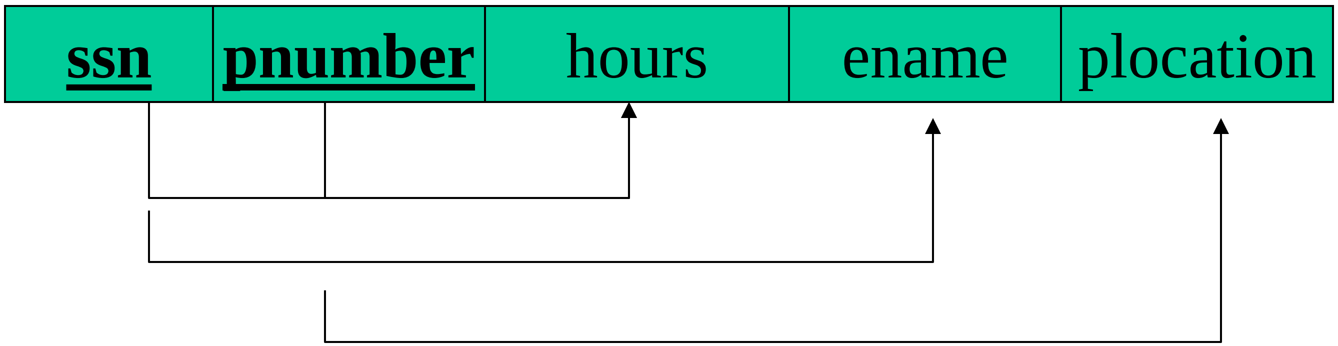

*ssn* is unique … given a specific *ssn* there is only one *ename*, only one *bdate*, etc

$ssn \rightarrow ename,$
$ssn \rightarrow bdate,$
$ssn \rightarrow address,$
$ssn \rightarrow dnumber.$

# Normalization

Suppose we have the following relation

**EmployeeProject**



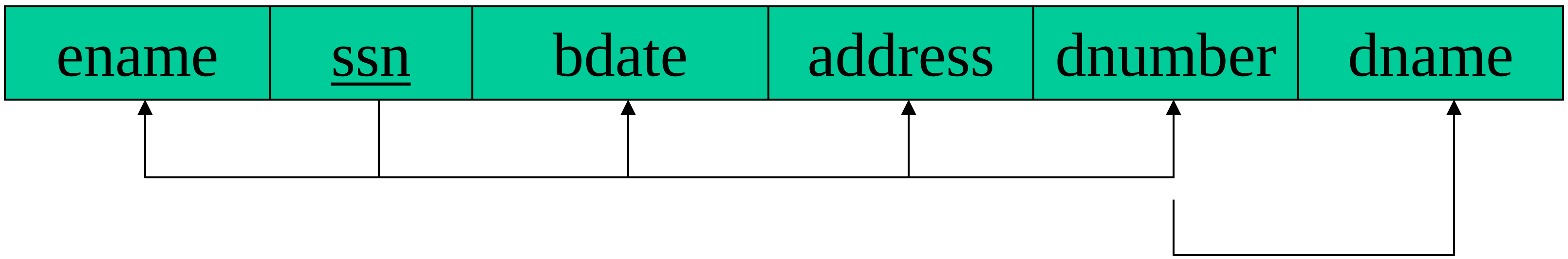

This is similar to *Works_on*, but we have included *ename*, and we know that *ename* is functionally dependent on *ssn.*

We have included *plocation* … functionally dependent on *pnumber*

{ssn, pnumber} → hours,
ssn → ename,
pnumber → plocation.

Suppose we have the following relation

**EmployeeDept**




This is similar to *Employee*, but we have included *dname*, and we know that *dname* is functionally dependent on *dnumber*, as well as being functionally dependent on *ssn*.

ssn → ename, ssn → bdate,
ssn → address, ssn → dnumber,
dnumber → dname. ssn → dname

**Minimal sets of FDs**

- every dependency has a single attribute on the RHS
- the attributes on the LHS of a dependency are minimal
- we cannot remove a dependency without losing information.

**Inference Rules for Function Dependencies**

- From a set of FDs, we can derive some other FDs

Example:

F = {ssn $\rightarrow$ {Ename, Bdate, Address, dnumber},
dnumber $\rightarrow$ {dname, dmgrssn}}

$\Downarrow$ *inference*

ssn $\rightarrow$ dnumber,
dnumber $\rightarrow$ dname.
ssn $\rightarrow$ {dname, dmgrssn},

- $F^+$ (closure of F): The set of all FDs that can be deduced from F (with F together) is called the closure of F.

## Inference Rules for Function Dependencies

- Inference rules:
  - IR1 (reflexive rule): If $X \supseteq Y$, then $X \rightarrow Y$. ($X \rightarrow X$.)
  - IR2 (augmentation rule): $\{X \rightarrow Y\} \models ZX \rightarrow ZY$.
  - IR3 (transitive rule): $\{X \rightarrow Y, Y \rightarrow Z\} \models X \rightarrow Z$.
  - IR4 (decomposition, or projective, rule):

    $$\{X \rightarrow ZY\} \models X \rightarrow Y, X \rightarrow Z.$$
  - IR5 (union, or additive, rule): $\{X \rightarrow Y, Y \rightarrow Z\} \models X \rightarrow ZY$.
  - IR6 (pseudotransitive rule): $\{X \rightarrow Y, WY \rightarrow Z\} \models WX \rightarrow Z$.

**Equivalence of Sets of FDs**

**E and F are equivalent if $E^+ = F^+$.**

**Minimal sets of FDs**

- Every dependency has a single attribute on the RHS
- The attributes on the LHS of a dependency are minimal
- We cannot remove any dependency from F and still have a set of dependencies that is equivalent to F.

| <u>**ssn**</u> | <u>**pnumber**</u> | hours | ename | plocation |
|---|---|---|---|---|

{ssn, pnumber} $\rightarrow$ hours,
ssn $\rightarrow$ ename,
pnumber $\rightarrow$ plocation.

## Normal Forms

- A series of normal forms are known that have, successively, better update characteristics.

- We'll consider 1NF, 2NF, 3NF, and BCNF.

- A technique used to improve a relation is decomposition, where one relation is replaced by two or more relations. When we do so, we want to eliminate update anomalies without losing any information.

## 1NF - First Normal Form

The domain of an attribute must only contain atomic values.

•This disallows repeating values, sets of values, relations within relations, nested relations, …

•In the example database we have a department located in possibly several locations: department 5 is located in Bellaire, Sugarland, and Houston.

•If we had the relation

**Department**

| dnumber | dname | dmgrssn | dlocations |
|---|---|---|---|
| 5 | Research | 333445555 | Bellaire, Sugarland, Houston |

then it would not be 1NF because there are multiple values to be kept in *dlocations*.

## 1NF - First Normal Form

If we have a non-1NF relation we can *decompose* it, or modify it appropriately, to generate 1NF relations.

There are 3 options:

•**option 1**: split off the problem attribute into a new relation (create a DepartmentLocation relation).

**Department**

| <u>dnumber</u> | dname | dmgrssn |
|---|---|---|
| 5 | Research | 333445555 |

**DepartmentLocation**

| <u>dnumber</u> | <u>dlocation</u> |
|---|---|
| 5 | Bellaire |
| 5 | Sugarland |
| 5 | Houston |

*Generally considered the best solution*

## 1NF - First Normal Form

•**option 2**: store just one value in the problem attribute, but create additional rows so that the other values can be stored too (department 5 would have 3 rows)

*Dlocation becomes part of PK*

**Department**

| dnumber | dname | dmgrssn | dlocation |
|---|---|---|---|
| 5 | Research | 333445555 | Bellaire |
| 5 | Research | 333445555 | Sugarland |
| 5 | Research | 333445555 | Houston |

**Redundancy is introduced!**

**(not in 2NF)**

## 1NF - First Normal Form

•**option 3**: if a maximum number of values is known, then create additional attributes so that the maximum number of values can be stored. (each location attribute would hold one location only)

**Department**

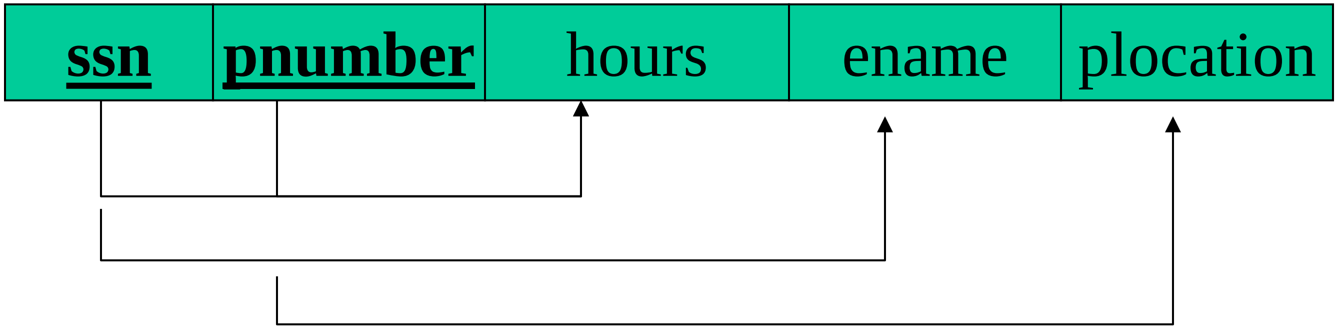
| <u>dnumber</u> | dname | dmgrssn | dloc1 | dloc2 | dloc3 |
|---|---|---|---|---|---|
| 5 | Research | 333445555 | Bellaire | Sugarland | Houston |

## 2NF - Second Normal Form

- full functional dependency

$X \rightarrow Y$ is a full functional dependency if removal of any attribute A from X means that the dependency does not hold any more.

**EmployeeProject**



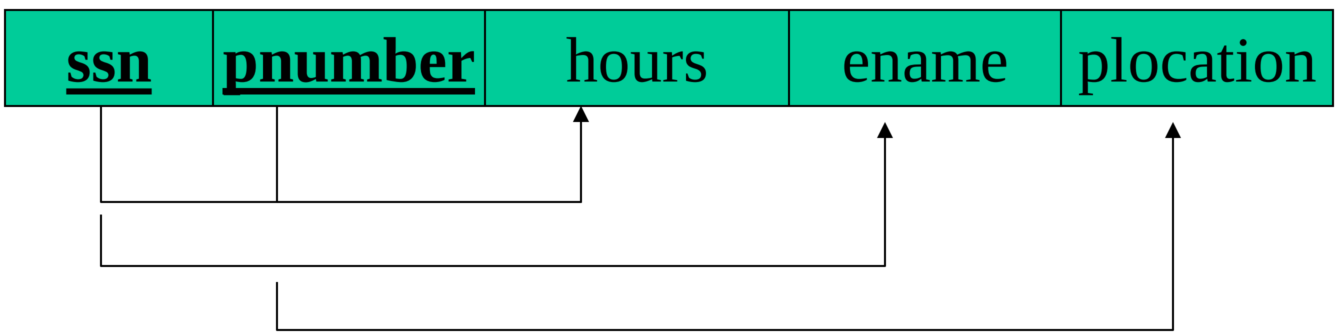

{*ssn*, *pnumber*} $\rightarrow$ *hours* is a full dependency (neither *ssn* $\rightarrow$ *hours* , nor *pnumber* $\rightarrow$ *hours*).

## 2NF - Second Normal Form

- partial functional dependency

  $X \rightarrow Y$ is a partial functional dependency if removal of some attribute A from X does not affect the dependency.

**EmployeeProject**




{*ssn*, *pnumber*} $\rightarrow$ *ename* is a partial dependency because *ssn* $\rightarrow$ *ename* holds.)

**2NF - Second Normal Form**

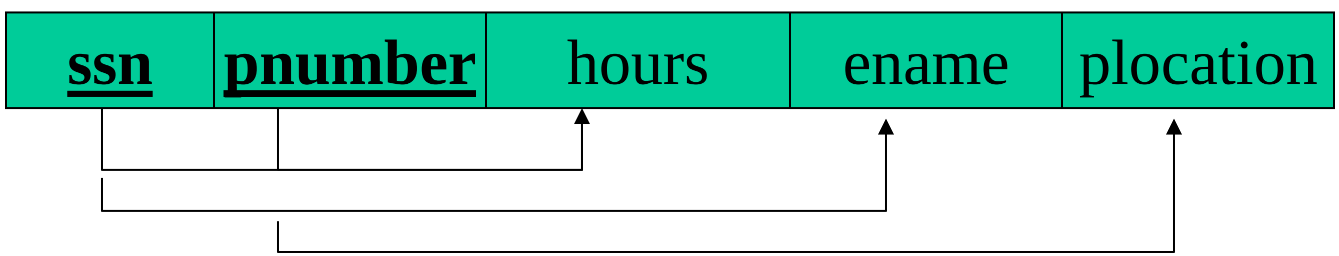
A relation schema is in 2NF if

(1) it is in 1NF and

(2) every non-key attribute must be fully functionally dependent on the candidate key.

If we had the relation

**EmployeeProject**




then this relation would not be 2NF because of two separate violations of the 2NF definition:

# Normalization

- *ename* is functionally dependent on *ssn*, and
- *plocation* is functionally dependent on *pnumber*

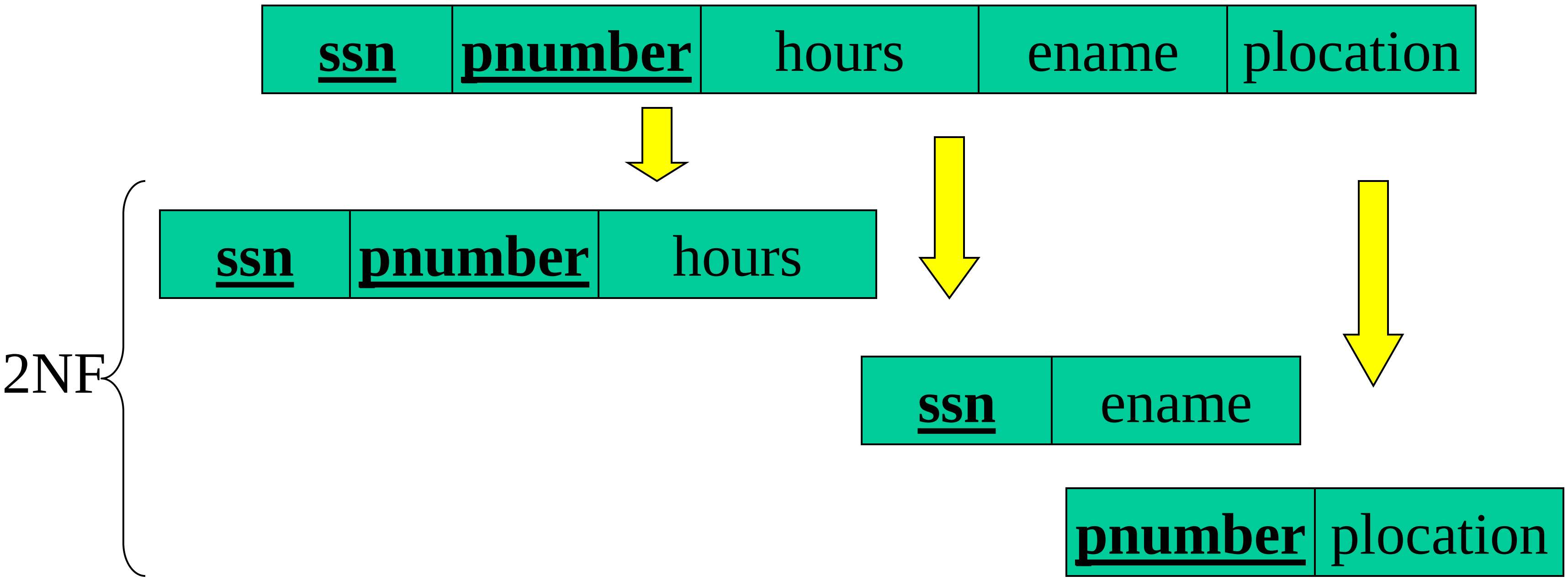
- *ename* is not fully functionally dependent on *ssn* and *pnumber* and
- *plocation* is not fully functionally dependent on *ssn* and *pnumber.*

{ssn, pnumber} is the primary key of EmployeeProject.

## 2NF - Second Normal Form

- We correct this by decomposing the relation into three relations - splitting off the offending attributes - splitting off partial dependencies on the key.

**EmployeeProject**

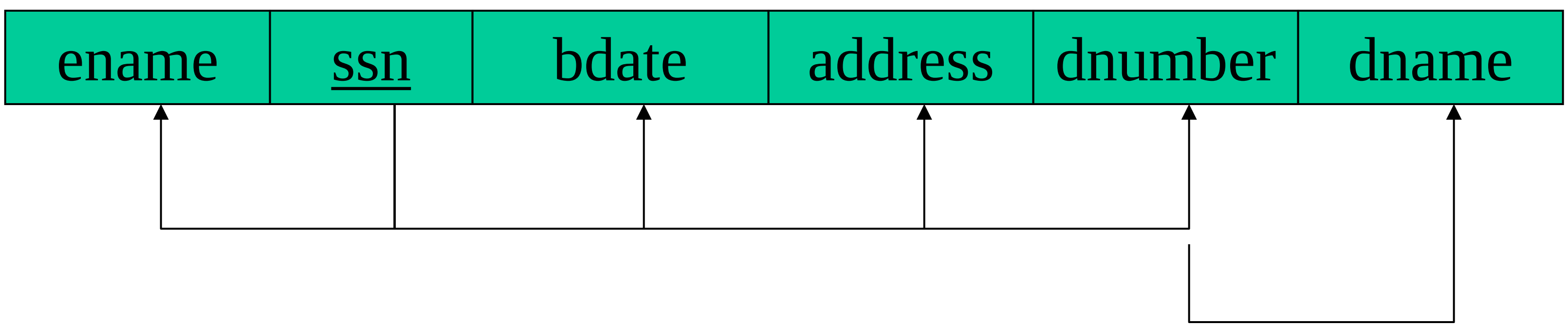


## 3NF - Third Normal Form

- Transitive dependency

A functional dependency $X \rightarrow Y$ in a relation schema R is a transitive dependency if there is a set of attributes Z that is not a subset of any candidate key of R, and both $X \rightarrow Z$ and $Z \rightarrow Y$ hold.

**EmployeeDept**




*ssn* $\rightarrow$ *dnumber* and *dnumber* $\rightarrow$ *dname*

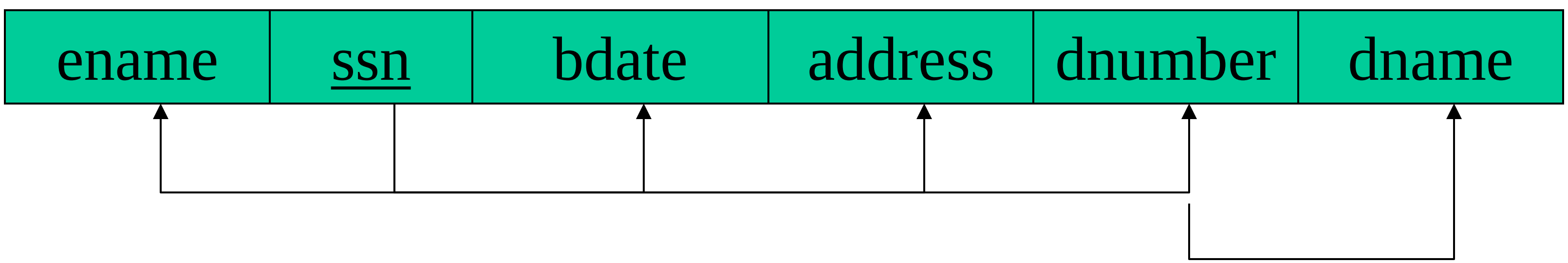
## 3NF - Third Normal Form

A relation schema is in 3NF if

(1) it is in 2NF and

(2) each non-key attribute must not be fully functionally dependent on another non-key attribute (there must be no transitive dependency of a non-key attribute on any candidate key.)

If we had the relation




then this relation would not be 3NF because

- *dname* is functionally dependent on *dnumber* and neither is
- a key attribute

## 3NF - Third Normal Form

- We correct this by decomposing - splitting off the transitive dependencies

**EmployeeDept**

| ename | ssn | bdate | address | dnumber | dname |
|---|---|---|---|---|---|

3NF

| ename | ssn | bdate | address | dnumber |
|---|---|---|---|---|

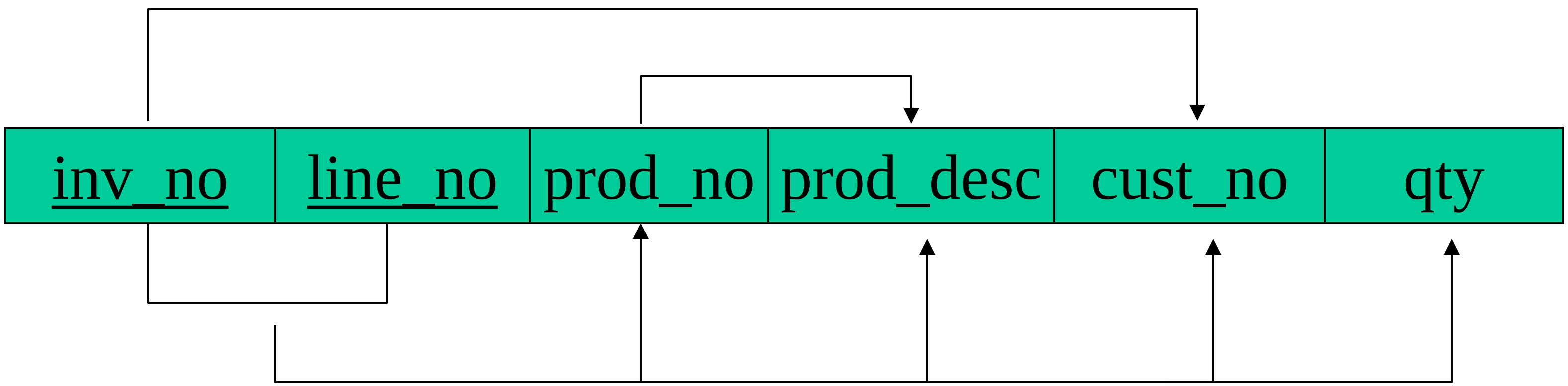
| dnumber | dname |
|---|---|

# Normalization

**Consider:**

What normal form is it in?

What relations will decomposition result in?




{inv_no, line_no} $\rightarrow$ prod_no,
{inv_no, line_no} $\rightarrow$ prod_desc,
{inv_no, line_no} $\rightarrow$ cust_no,
{inv_no, line_no} $\rightarrow$ qty,
inv_no $\rightarrow$ cust_no, prod_no $\rightarrow$ prod_desc

inv_no: invoice number
line_no: invoice line number

**Change it into 2NF:**

| inv_no | line_no | prod_no | prod_desc | cust_no | qty |
|---|---|---|---|---|---|

↓

2NF

| inv_no | line_no | prod_no | prod_desc | qty |
|---|---|---|---|---|

| inv_no | cust_no |
|---|---|

## Change it into 3NF:

2NF

| inv_no | line_no | prod_no | prod_desc | qty |
|---|---|---|---|---|

| inv_no | cust_no |
|---|---|

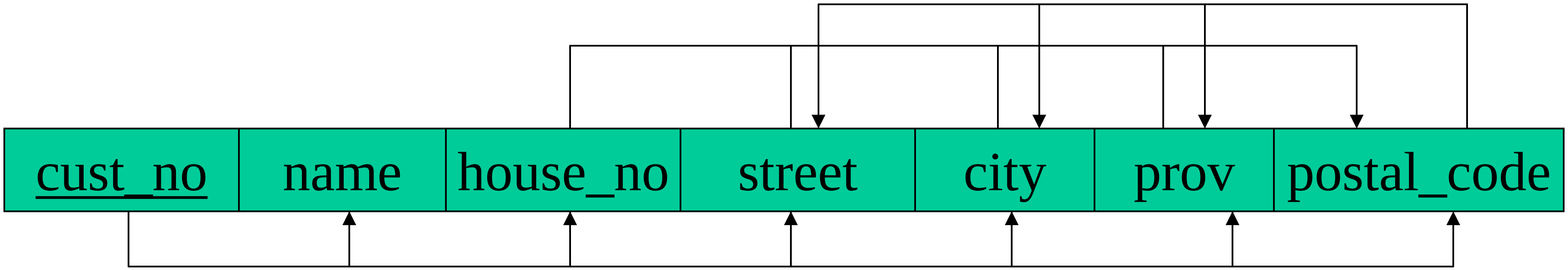
3NF

| inv_no | line_no | prod_no | qty |
|---|---|---|---|

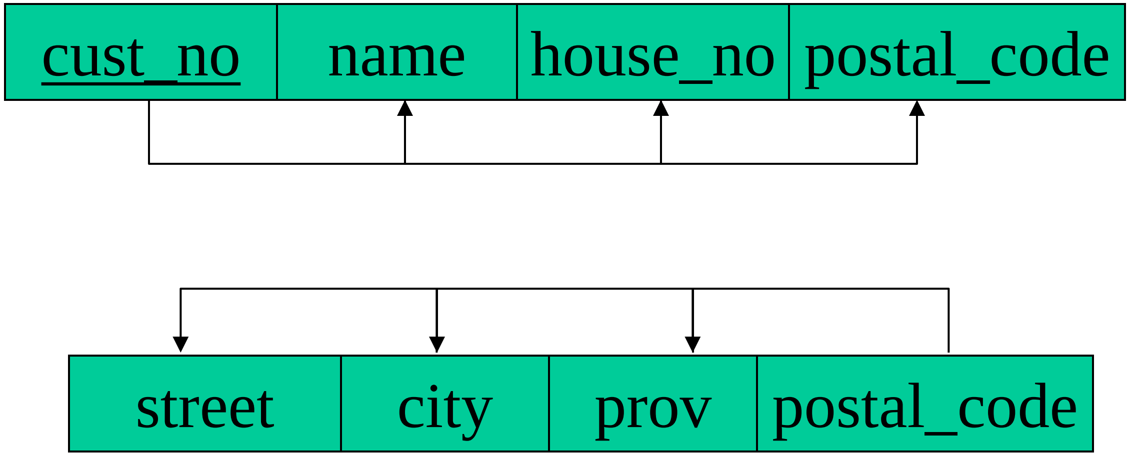
| inv_no | cust_no |
|---|---|

| prod_no | prod_desc |
|---|---|

**Consider:**



# Normalization



**Boyce Codd Normal Form**, BCNF

- Consider a different definition of 3NF, which is equivalent to the previous one.

  A relation schema R is in 3NF if, whenever a function dependency $X \rightarrow A$ holds in R, either

  (a) X is a superkey of R, or

  (b) A is a prime attribute of R.

A superkey of a relation schema R = {A1, A2, ..., An} is a set of attributes $S \subseteq R$ with the propertity that no tuples t1 and t2 in any legal state r of R will have t1[S] = t2[S].
An attribute is called a prime attribute if it is a member of any key.

**Boyce Codd Normal Form**, BCNF

•Consider a different definition of 3NF, which is equivalent to the previous one.

> A relation schema R is in 3NF if, whenever a function dependency $X \rightarrow A$ holds in R, either
>
> (a) X is a superkey of R, or
>
> (b) A is a prime attribute of R.

There is no non-key attribute Y partially depends on a key X.
There is no non-key attribute Y transitively depends on a key X.

(A functional dependency $X \rightarrow Y$ in a relation schema R is a transitive dependency if there is a set of attributes Z that is not a subset of any key of R, and both $X \rightarrow Z$ and $Z \rightarrow Y$ hold.)

**Boyce Codd Normal Form**, BCNF

•If we remove (b) from the previous definition for 3NF, we have the definition for BCNF.

•A relation schema is in BCNF if every determinant is a superkey key. Stronger than 3NF:

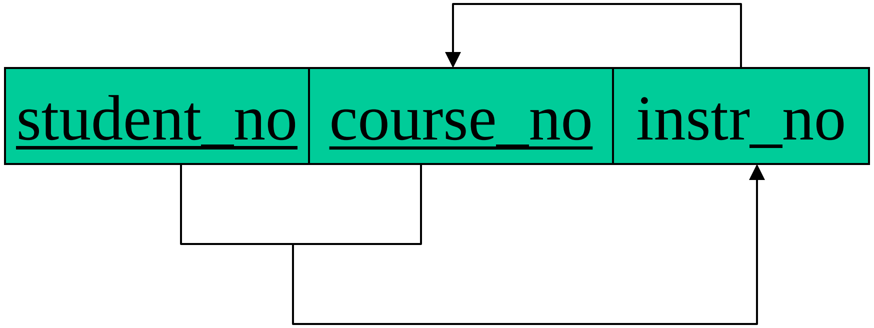
- no partial dependencies

- no transitive dependencies where a non-key attribute is dependent on another non-key attribute

- no non-key attributes appear in the LHS of a functional dependency.

**Boyce Codd Normal Form**, BCNF

Consider:




*Instructor teaches one course only.*

**In 3NF!**

*Student takes a course and has one instructor.*

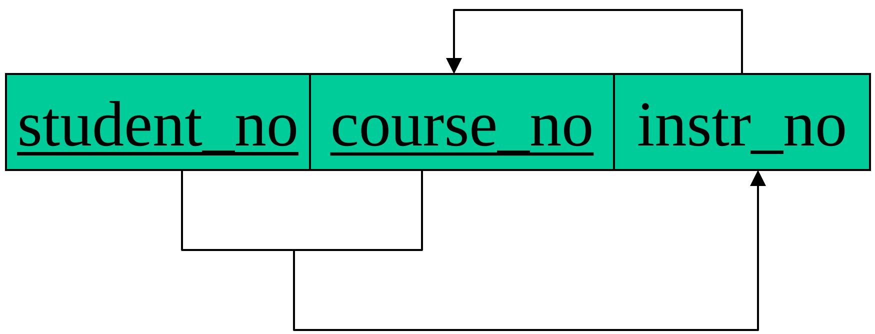
{student_no, course_no} → instr_no

instr_no → course_no

# Normalization

**Boyce Codd Normal Form**, BCNF

Some sample data:




| | | |
|---|---|---|
| 121 | 1803 | 99 |
| 121 | 1903 | 77 |
| 222 | 1803 | 66 |
| 222 | 1903 | 77 |

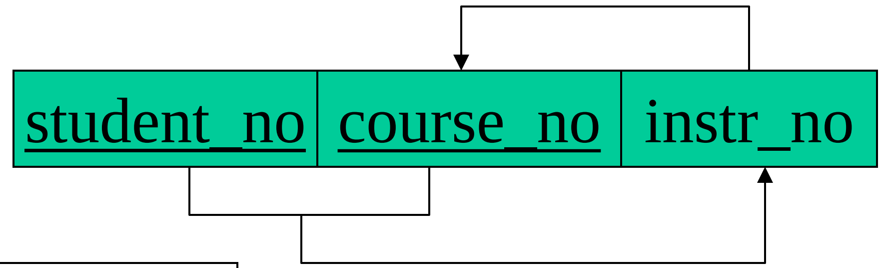
Instructor 99 teaches 1803

Instructor 77 teaches 1903

Instructor 66 teaches 1803

# Normalization

**Boyce Codd Normal Form**, BCNF




| | | |
|---|---|---|
| 121 | 1803 | 99 |
| 121 | 1903 | 77 |
| 222 | 1803 | 66 |
| 222 | 1903 | 77 |

Instructor 99 teaches 1803

Instructor 77 teaches 1903

Instructor 66 teaches 1803

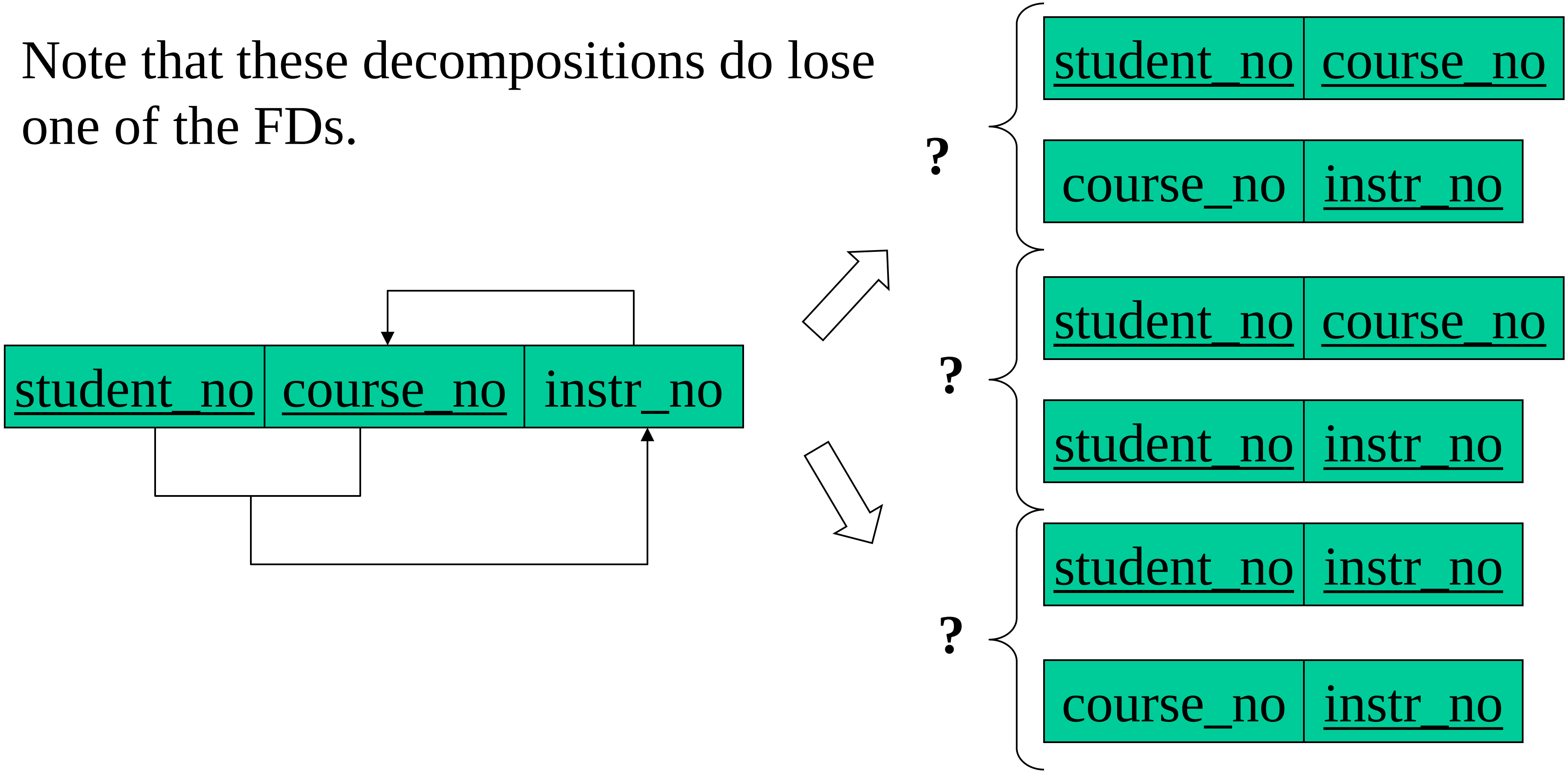
**Deletion anomaly:** If we delete all rows for course 1803 we'll lose the information that instructors 99 teaches student 121 and 66 teaches student 222.

**Insertion anomaly:** How do we add the fact that instructor 55 teaches course 2906?

**Boyce Codd Normal Form**, BCNF

How do we decompose this to remove the redundancies? - without losing information?

Note that these decompositions do lose one of the FDs.



**Boyce Codd Normal Form**, BCNF

Which decomposition preserves all the information?

| S# | C# |
|---|---|
| 121 | 1803 |
| 121 | 1903 |
| 222 | 1803 |
| 222 | 1903 |

| C# | I# |
|---|---|
| 1803 | 99 |
| 1903 | 77 |
| 1803 | 66 |

**?**

| student_no | course_no |
|---|---|

| course_no | instr_no |
|---|---|

| student_no | course_no |
|---|---|

| student_no | instr_no |
|---|---|

| student_no | instr_no |
|---|---|

| course_no | instr_no |
|---|---|

Joining these two tables leads to *spurious* tuples - result includes

121 1803 66

222 1803 99

# Normalization

| student_no | course_no | instr_no |
|---|---|---|
| 121 | 1803 | 99 |
| 121 | 1903 | 77 |
| 222 | 1803 | 66 |
| 222 | 1903 | 77 |

| student_no | course_no |
|---|---|
| S# | C# |
| 121 | 1803 |
| 121 | 1903 |
| 222 | 1803 |
| 222 | 1903 |

⋈

| course_no | instr_no |
|---|---|
| C# | I# |
| 1803 | 99 |
| 1903 | 77 |
| 1803 | 66 |

×

**Boyce Codd Normal Form**, BCNF

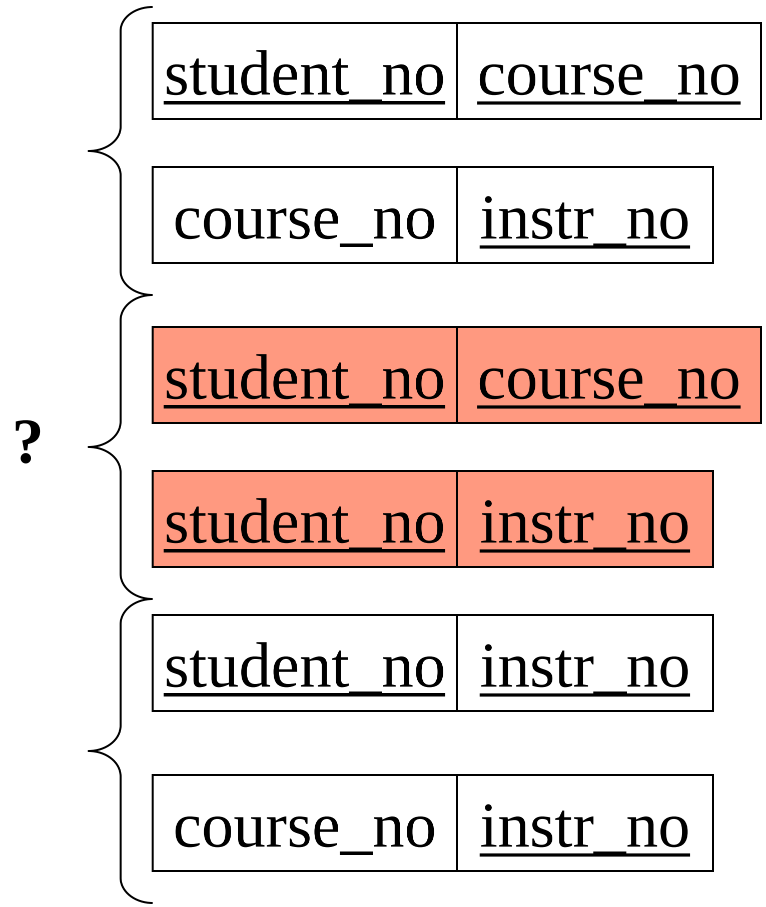
Which decomposition preserves all the information?

| S# | C# |
|---|---|
| 121 | 1803 |
| 121 | 1903 |
| 222 | 1803 |
| 222 | 1903 |

| S# | I# |
|---|---|
| 121 | 99 |
| 121 | 77 |
| 222 | 66 |
| 222 | 77 |

Joining these two tables leads to *spurious* tuples - result includes

121 1803 77
121 1903 99
222 1803 77
222 1903 66



# Normalization

| student_no | course_no | instr_no |
|---|---|---|

| | | |
|---|---|---|
| 121 | 1803 | 99 |
| 121 | 1903 | 77 |
| 222 | 1803 | 66 |
| 222 | 1903 | 77 |

| student_no | course_no |
|---|---|

| student_no | instr_no |
|---|---|

| S# | C# |
|---|---|
| 121 | 1803 |
| 121 | 1903 |
| 222 | 1803 |
| 222 | 1903 |

$\bowtie$

| S# | I# |
|---|---|
| 121 | 99 |
| 121 | 77 |
| 222 | 66 |
| 222 | 77 |

×

**Boyce Codd Normal Form**, BCNF

Which decomposition preserves all the information?

| S# | I# |
|---|---|
| 121 | 99 |
| 121 | 77 |
| 222 | 66 |
| 222 | 77 |

| C# | I# |
|---|---|
| 1803 | 99 |
| 1903 | 77 |
| 1803 | 66 |

Joining these two tables leads to **no** *spurious* tuples - result is:

121 1803 99
121 1903 77
222 1803 66
222 1903 77

| student_no | course_no |
|---|---|

| course_no | instr_no |
|---|---|

| student_no | course_no |
|---|---|

| student_no | instr_no |
|---|---|

**?**

| student_no | instr_no |
|---|---|

| course_no | instr_no |
|---|---|

**Boyce Codd Normal Form**, BCNF

This decomposition preserves all the information.

| S# | I# |
|---|---|
| 121 | 99 |
| 121 | 77 |
| 222 | 66 |
| 222 | 77 |

| C# | I# |
|---|---|
| 1803 | 99 |
| 1903 | 77 |
| 1803 | 66 |

| student_no | instr_no |
|---|---|

| course_no | instr_no |
|---|---|

Only FD is instr_no ⟶ course_no

but the join preserves

{student_no, course_no} ⟶ instr_no

# Normalization

| student_no | course_no | instr_no |
|---|---|---|

| | | |
|---|---|---|
| 121 | 1803 | 99 |
| 121 | 1903 | 77 |
| 222 | 1803 | 66 |
| 222 | 1903 | 77 |

| student_no | Instr_no |
|---|---|

| course_no | instr_no |
|---|---|

| S# | I# |
|---|---|
| 121 | 99 |
| 121 | 77 |
| 222 | 66 |
| 222 | 77 |

$\bowtie$

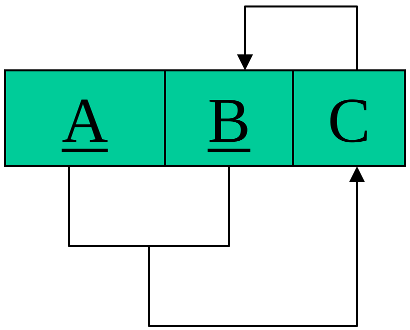
| C# | I# |
|---|---|
| 1803 | 99 |
| 1903 | 77 |
| 1803 | 66 |

**Boyce Codd Normal Form**, BCNF

A relation schema is in BCNF if every determinant is a candidate key.

**Boyce Codd Normal Form**, BCNF

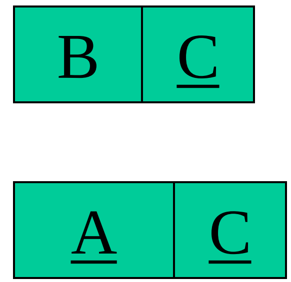
Given:




**In 3NF**

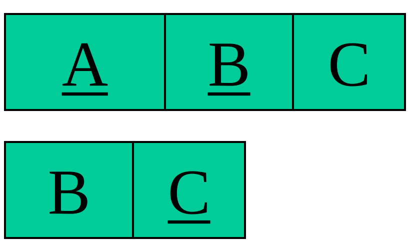
**Not in BCNF**

Lossless decomposition pattern:




**In BCNF**

**But** this could be where a database designer may decide to go with:




- *Functional dependencies are preserved*
- *There is some redundancy*
- *Delete anomaly is avoided*

# Outline: Lossless-join

- Basic definition of Lossless-join
- Examples
- Testing algorithm

- Basic definition of Lossless-join

A decomposition $D = \{R_1, R_2, \ldots, R_m\}$ of R has the *lossless join property* with respect to the set of dependencies F on R if, for every relation r of R that satisfies F, the following holds,

$$*(\pi_{R1}(r), \ldots, \pi_{Rm}(r)) = r,$$

where $*$ is the natural join of all the relations in D.

The word loss in lossless refers to *loss of information*, not to loss of tuples.

- Example: decomposition-1

**Emp_PROJ**

| SSN | PNUM | hours | ENAME | PNAME | PLOCATION |
|---|---|---|---|---|---|

F = {SSN → ENAME, PNUM → {PNAME, PLOCATION},
{SSN, PNUM} → hours}

⇓

R1

| SSN | ENAME |
|---|---|

R2

| PNUM | PNAME | PLOCATION |
|---|---|---|

R3

| SSN | PNUM | hours |
|---|---|---|

Lossless join

- Example: decomposition-2

**Emp_PROJ**

| SSN | PNUM | hours | ENAME | PNAME | PLOCATION |
|---|---|---|---|---|---|

F = {SSN → ENAME, PNUM → {PNAME, PLOCATION},
{SSN, PNUM} → hours}

⇓

R1

| ENAME | PLOCATION |
|---|---|

← Not lossless join

R2

| SSN | PNUM | hours | PNAME | PLOCATION |
|---|---|---|---|---|

# Normalization

- decomposition-1

| | A1 SSN | A2 ENAME | A3 PNUM | A4 PNAME | A5 PLOCATION | A6 hours |
|---|---|---|---|---|---|---|
| R1 | b11 | b12 | b13 | b14 | b15 | b16 |
| R2 | b21 | b22 | b23 | b24 | b25 | b26 |
| R3 | b31 | b32 | b33 | b34 | b35 | b36 |

⇓

| | | | | | | |
|---|---|---|---|---|---|---|
| R1 | a1 | a2 | b13 | b14 | b15 | b16 |
| R2 | b21 | b22 | a3 | a4 | a5 | b26 |
| R3 | a1 | b32 | a3 | b34 | b35 | a6 |

⇒

# Normalization

$SSN \rightarrow ENAME$

| | SSN | ENAME | | | | |
|---|---|---|---|---|---|---|
| R1 | a1 | a2 | b13 | b14 | b15 | b16 |
| R2 | b21 | b22 | a3 | a4 | a5 | b26 |
| R3 | a1 | a2 | a3 | b34 | b35 | a6 |

$PNUM \rightarrow \{PNAME, PLOCATION\}$

| | | | PNUM | PNAME | PLOCATION | |
|---|---|---|---|---|---|---|
| R1 | a1 | a2 | b13 | b14 | b15 | b16 |
| R2 | b21 | b22 | a3 | a4 | a5 | b26 |
| R3 | a1 | a2 | a3 | a4 | a5 | a6 |

- decomposition-2

| | A1 SSN | A2 ENAME | A3 PNUM | A4 PNAME | A5 PLOCATION | A6 hours |
|---|---|---|---|---|---|---|
| R1 | b11 | b12 | b13 | b14 | b15 | b16 |
| R2 | b21 | b22 | b23 | b24 | b25 | b26 |

⇓

| | | | | | | |
|---|---|---|---|---|---|---|
| R1 | b11 | a2 | b13 | b14 | a5 | b16 |
| R2 | a1 | b22 | a3 | a4 | a5 | a6 |

⇓

SSN → ENAME
PNUM → {PNAME, PLOCATION}
{SSN, PNUM} → hours

The matrix can not be changed!

Why?

Decomposition-1:

EMP_PROJ

| | | | | | |
|---|---|---|---|---|---|
| a1 | a2 | b13 | b14 | b15 | b16 |
| b21 | b22 | a3 | a4 | a5 | a6 |
| a1 | a2 | a3 | a4 | a5 | a6 |

R1

| | |
|---|---|
| a1 | a2 |
| b21 | b22 |

R2

| | | |
|---|---|---|
| b13 | b14 | b15 |
| a3 | a4 | a5 |

R3

| | | |
|---|---|---|
| a1 | b13 | b16 |
| b21 | a3 | b26 |
| a1 | a3 | a6 |

R1

| SSN | ENAME |
|---|---|

R2

| PNUM | PNAME | PLOCATION |
|---|---|---|

R3

| SSN | PNUM | hours |
|---|---|---|

Why?

Decomposition-1:

R1 * R3 = R13 =

| | | | |
|---|---|---|---|
| a1 | a2 | b13 | b16 |
| a1 | a2 | a3 | a6 |
| b21 | b22 | a3 | b26 |

R13 * R2 =

| | | | | | |
|---|---|---|---|---|---|
| a1 | a2 | b13 | b14 | b15 | b16 |
| b21 | b22 | a3 | a4 | a5 | a6 |
| a1 | a2 | a3 | a4 | a5 | a6 |

Why?

Decomposition-2:

EMP_PROJ

| | | | | | |
|---|---|---|---|---|---|
| b11 | a2 | b13 | b14 | a5 | b16 |
| a1 | b22 | a3 | a4 | a5 | a6 |

R1

| ENAME | PLOCATION |
|---|---|

R2

| SSN | PNUM | hours | PNAME | PLOCATION |
|---|---|---|---|---|

R1

| | |
|---|---|
| a2 | a5 |
| b22 | a5 |

R2

| | | | | |
|---|---|---|---|---|
| b11 | b13 | b14 | a5 | b16 |
| a1 | a3 | a4 | a5 | a6 |

Why?

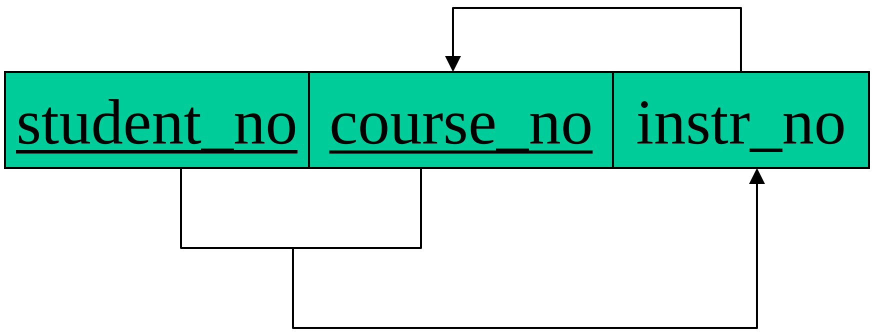
Decomposition-2:

R1 * R2 =

| | | | | | |
|---|---|---|---|---|---|
| b11 | a2 | b13 | b14 | a5 | b16 |
| a1 | a2 | a3 | a4 | a5 | a6 |
| b11 | b22 | b13 | b14 | a5 | b16 |
| a1 | b22 | a3 | a4 | a5 | a6 |

Spurious tuples

Student-course-instructor:




*Instructor's teach one course only*

*Student takes a course and has one instructor*

{student_no, course} $\rightarrow$ instr_no

instr_no $\rightarrow$ course_no

# Normalization

| | | |
|---|---|---|
| student_no | course_no | instr_no |

R1

| | |
|---|---|
| Course_no | instr_no |

R2

| | |
|---|---|
| student_no | instr_no |

$\{student\_no, course\} \rightarrow instr\_no$

$instr\_no \rightarrow course\_no$

| | A1<br>stu-no | A2<br>course-no | A3<br>instr-no |
|---|---|---|---|
| R1 | b11 | b12 | b13 |
| R2 | b21 | b22 | b23 |

⇒

| | A1<br>stu-no | A2<br>course-no | A3<br>instr-no |
|---|---|---|---|
| R1 | b11 | a2 | a3 |
| R2 | a1 | b22 | a3 |

⇒

| | | | |
|---|---|---|---|
| R1 | b11 | a2 | a3 |
| R2 | a1 | a2 | a3 |

# Normalization

| student_no | course_no | instr_no |
|---|---|---|

R1

| Course_no | instr_no |
|---|---|

R2

| student_no | course_no |
|---|---|

{student_no, course} $\rightarrow$ instr_no

instr_no $\rightarrow$ course_no

| | A1<br>stu-no | A2<br>course-no | A3<br>instr-no |
|---|---|---|---|
| R1 | b11 | b12 | b13 |
| R2 | b21 | b22 | b23 |

$\Rightarrow$

| | A1<br>stu-no | A2<br>course-no | A3<br>instr-no |
|---|---|---|---|
| R1 | b11 | a2 | a3 |
| R2 | a1 | a2 | b23 |

instr_no $\rightarrow$ course_no

| | | | |
|---|---|---|---|
| R1 | b11 | a2 | a3 |
| R2 | a1 | a2 | b23 |

# Normalization

| student_no | course_no | instr_no |
|---|---|---|

R1

| student_no | instr_no |
|---|---|

R2

| student_no | course_no |
|---|---|

{student_no, course} $\rightarrow$ instr_no

instr_no $\rightarrow$ course_no

| | A1<br>stu-no | A2<br>course-no | A3<br>instr-no |
|---|---|---|---|
| R1 | b11 | b12 | b13 |
| R2 | b21 | b22 | b23 |

$\Rightarrow$

| | A1<br>stu-no | A2<br>course-no | A3<br>instr-no |
|---|---|---|---|
| R1 | a1 | b12 | a3 |
| R2 | a1 | a2 | b23 |

$\Rightarrow$

| | | | |
|---|---|---|---|
| R1 | a1 | b12 | a3 |
| R2 | a1 | a2 | b23 |

**Testing algorithm**

input: A relation R, a decomposition D = {$R_1$, $R_2$,..., $R_m$} of R, and a set F of function dependencies.

1. Create an initial matrix S with one row *i* for each relation R*i* in D, and one column *j* for each attribute A*j* in R.
2. Set $S(i, j) := b_{ij}$ for all matrix entries.
3. **For** each row *i* representing relation schema R*i* **Do**
   {**for** each column *j* representing A*j* **do**
   {**if** relation R*i* includes attribute A*j* **then**
   set $S(i, j) := a_j$;}
4. Repeat the following loop until a complete loop execution results in no changes to S.

4. Repeat the following loop until a complete loop execution results in no changes to S.

{**for** each function dependency X → Y in F **do**

**for** all rows in S which have the same symbols in the columns corresponding to attributes in X **do**

{make the symbols in each column that correspond to an attribute in Y be the same in all these 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. If no “a” symbol exists for the attribute in any of the rows, choose one of the “b” symbols that appear in one of the rows for the attribute and set the other rows to that same “b” symbol in the column;}}

5. If a row is made up entirely of “a” symbols, then the decomposition has the lossless join property; otherwise it does not.

| | | | | | |
|---|---|---|---|---|---|
| a1 | a2 | b13 | b14 | b15 | b16 |
| b21 | b22 | a3 | a4 | a5 | b26 |
| a1 | b32 | a3 | b34 | b35 | a6 |

R1<SSN, ENAME>

| | |
|---|---|
| a1 | a2 |
| b21 | b22 |
| a1 | b32 |

R2<PNUM, PNAME, Plocation>

| | | |
|---|---|---|
| b13 | b14 | b15 |
| a3 | a4 | a5 |
| a3 | b34 | b35 |

R3<SSN, PNUM, hours>

| | | |
|---|---|---|
| a1 | b13 | b16 |
| b21 | a3 | b26 |
| a1 | a3 | a6 |

PNUM → {PNAME, PLOCATION}

<a3, a4, a5, a1, a3, a6>

<a3, b34, b35, a1, a3, a6>