Schema Refinement and Normal Forms

Problems caused by redundancy

Id	name	lot	rating	Hourly_wages	Hours_worked
123-22-3666	Attishoo	48	8	10	40
231-31-5368	Smiley	22	8	10	30
131-24-3650	Smethurst	35	5	7	30
434-26-3751	Guldu	35	5	7	32
612-67-4134	Madayan	35	8	10	40

Redundant Storage

The rating value 8 corresponds to the hourly wage 10, and this association is repeated three times.

Update Anomalies

The hourly_wages in the first tuple could be updated without making a similar change in the second tuple.

Id	name	lot	rating	Hourly_wages	Hours_worked
123-22-3666	Attishoo	48	8	10	40
231-31-5368	Smiley	22	8	10	30
131-24-3650	Smethurst	35	5	7	30
434-26-3751	Guldu	35	5	7	32
612-67-4134	Madayan	35	8	10	40

Insertion Anomalies

We cannot insert a tuple for an employee unless we know the hourly wage for the employee's rating value.

Deletion Anomalies

If we delete all tuples with a given rating value (e.g. tuples of Smethurst and Guldu) we lose the association between the rating value and its hourly_wage value.

Data Redundancy and Update Anomalies

Staff

staffNo	sName	position	salary	branchNo
SL21	John White	Manager	30000	B005
SG37	Ann Beech	Assistant	12000	B003
SG14	David Ford	Supervisor	18000	B003
SA9	Mary Howe	Assistant	9000	B007
SG5	Susan Brand	Manager	24000	B003
SL41	Julie Lee	Assistant	9000	B005

Branch

branchNo	bAddress
B005	22 Deer Rd, London
B007	16 Argyll St, Aberdeen
B003	163 Main St, Glasgow

Staff Branch

staffNo	sName	position	salary	branchNo	bAddress
SL21	John White	Manager	30000	B005	22 Deer Rd, London
SG37	Ann Beech	Assistant	12000	B003	163 Main St, Glasgow
SG14	David Ford	Supervisor	18000	B003	163 Main St, Glasgow
SA9	Mary Howe	Assistant	9000	B007	16 Argyll St, Aberdeen
SG5	Susan Brand	Manager	24000	B003	163 Main St, Glasgow
SL41	Julie Lee	Assistant	9000	B005	22 Deer Rd, London

Data Redundancy and Update Anomalies

- StaffBranch relation has redundant data; the details of a branch are repeated for every member of staff.

- In contrast, the branch information appears only once for each branch in the Branch relation and only the branch number (branchNo) is repeated in the Staff relation, to represent where each member of staff is located.

Data Redundancy and Update Anomalies

- Relations that contain redundant information may potentially suffer from update anomalies.

- Types of update anomalies include
  - Insertion
  - Deletion
  - Modification

Problems caused by redundancy

Redundant Storage

Some information is stored repeatedly.

Update Anomalies

If one copy of such repeated data is updated, an inconsistency is created, unless all copies are similarly updated.

Insertion anomalies

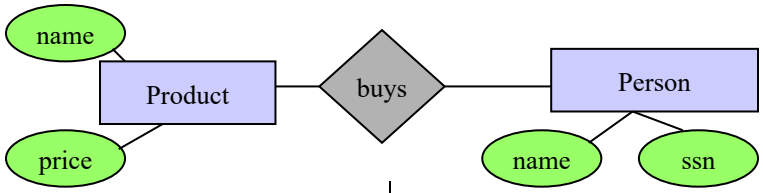
It may not be possible to store certain information unless some other, unrelated, information is stored.

Deletion Anomalies

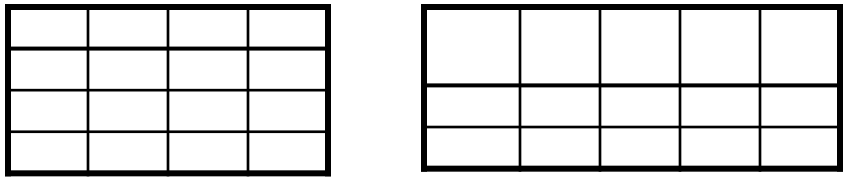
It may not be possible to delete certain information without losing some other, unrelated, information.

Relational Schema Design

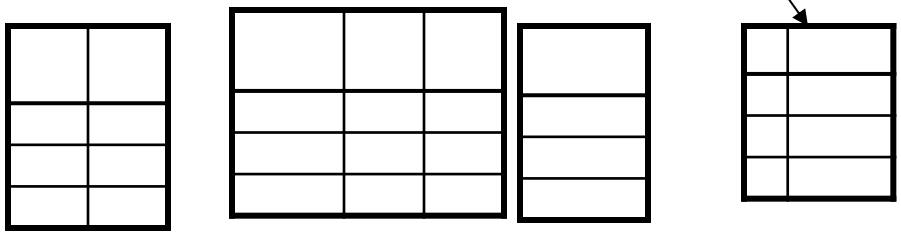
Conceptual Model:



Relational Model:
plus FD's



Normalization:
Eliminates anomalies



•Decompositions should always be lossless

Lossless decomposition ensure that the information in the original relation can be accurately reconstructed based on the information represented in the decomposed relations.

Decompositions

- Intuitively, redundancy arise when a relational schema forces an association between attributes that is not natural.
- Functional dependencies can be used to identify such situations and suggest refinements to the schema.
- The essential idea is that many problems arising from redundancy can be addressed by replacing a relation with a collection of 'smaller' relations.

rating is not a key here

Id	name	lot	rating	Hourly_wages	Hours_worked
123-22-3666	Attishoo	48	8	10	40
231-31-5368	Smiley	22	8	10	30
131-24-3650	Smethurst	35	5	7	30
434-26-3751	Guldu	35	5	7	32
612-67-4134	Madayan	35	8	10	40

Rating is key here

Id	name	lot	rating	Hours_worked
123-22-3666	Attishoo	48	8	40
231-31-5368	Smiley	22	8	30
131-24-3650	Smethurst	35	5	30
434-26-3751	Guldu	35	5	32
612-67-4134	Madayan	35	8	40

rating	Hourly_wages
8	10
5	7

Functional dependency:
-rating determines Hourly_wages
-In either case

A decomposition of a relation schema R consists of replacing the relation schema by two (or more) relation schemas so that each contains a subset of attributes of R and together include all attributes in R

Lossless-join and Dependency Preservation Properties

- **Two important properties of decomposition.**
  - ***Lossless-join property*** **enables us to find any instance of the original relation from corresponding instances in the smaller relations.**
  - ***Dependency preservation property*** **enables us to enforce a constraint on the original relation by enforcing some constraint on each of the smaller relations.**

Functional Dependencies

A functional dependency (FD) is a kind of Integrity Constraint that generalizes the concept of a key.

Let R be a relation schema and let X and Y be nonempty sets of attributes in R.

An instance (relation) r of R satisfies the FD X→ Y

if the following holds for every pair of tuples t1 and t2 in r

If t1.X = t2.X, then t1.Y = t2.Y

The notation t1.X refers to the projection of tuple t1 onto the attributes in X

An FD X→ Y essentially says that if two tuples agree on the values in attributes X, they must also agree on the values in attributes Y.

student_ID	student_name	course_ID	course_name
111	Chan Tai Man	3170	Database
222	Wong Siu Ling	3170	Database
333	Tam Wai Ming	3160	Algorithms
111	Chan Tai Man	3160	Algorithms

course_ID → course_name

student_ID course_ID → course_name

course_ID → course_ID course_name

course_ID course_name → course_ID course_name

Functional Dependencies

Let R be a relation schema and let X and Y be nonempty sets of attributes in R.
The function dependency X→ Y holds over R if
if for any instance r of R the following holds for every pair of tuples t1 and t2 in r

if t1.X = t2.X, then t1.Y = t2.Y

X→ Y : for every pair t1 and t2 if t1.X = t2.X, then t1.Y = t2.Y

student_ID	student_name	course_ID	course_name
111	Chan Tai Man	3170	Database
222	Wong Siu Ling	3170	Database
333	Tam Wai Ming	3160	Algorithms
111	Chan Tai Man	3160	Algorithms

if any two rows never agree on the α value, then $\alpha \rightarrow \beta$ is trivially preserved.

course_ID → course_name is not trivially preserved

student_ID, course_ID → course_name is trivially preserved
-- all (student_ID, course_ID) value pairs are unique

X→ Y : for every pair t1 and t2 if t1.X = t2.X, then t1.Y = t2.Y

student_ID	student_name	course_ID	course_name
111	Chan Tai Man	3170	Database
222	Wong Siu Ling	3170	Database
333	Tam Wai Ming	3160	Algorithms
111	Chan Tai Man	3160	Algorithms

Trivial dependency: LHS contains RHS

student_name → student_name (a trivial dependency)

student_name course_name → student_name (trivial *preserved and trivial dependency*)

student_ID course_ID →

student_ID student_name course_ID course_Name

(not a trivial dependency, but trivially preserved)

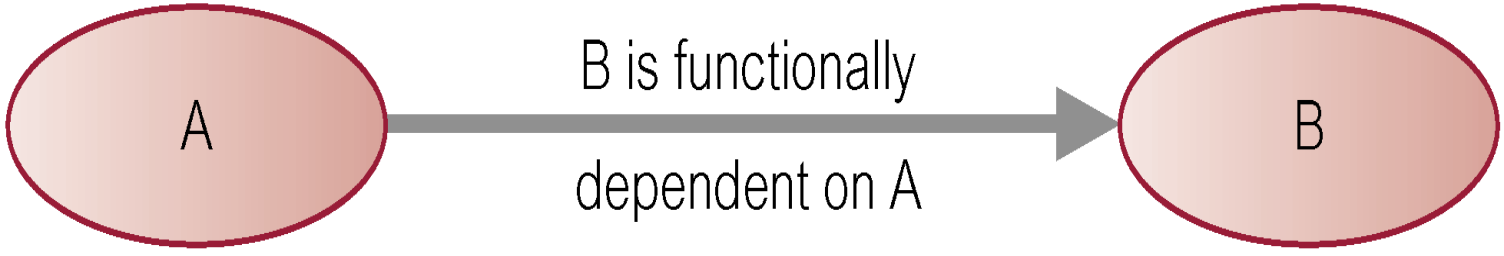
Functional Dependencies

- Important concept associated with normalization.
- Functional dependency describes relationship between attributes.
- For example, if A and B are attributes of relation R, B is functionally dependent on A (denoted A $\rightarrow$ B), if each value of A in R is associated with exactly one value of B in R.

Characteristics of Functional Dependencies

- **Property of the meaning or semantics of the attributes in a relation.**

- **Diagrammatic representation.**



- **The *determinant* of a functional dependency refers to the attribute or group of attributes on the left-hand side of the arrow.**

Sample Data

Staff

staffNo	sName	position	salary	branchNo
SL21	John White	Manager	30000	B005
SG37	Ann Beech	Assistant	12000	B003
SG14	David Ford	Supervisor	18000	B003
SA9	Mary Howe	Assistant	9000	B007
SG5	Susan Brand	Manager	24000	B003
SL41	Julie Lee	Assistant	9000	B005

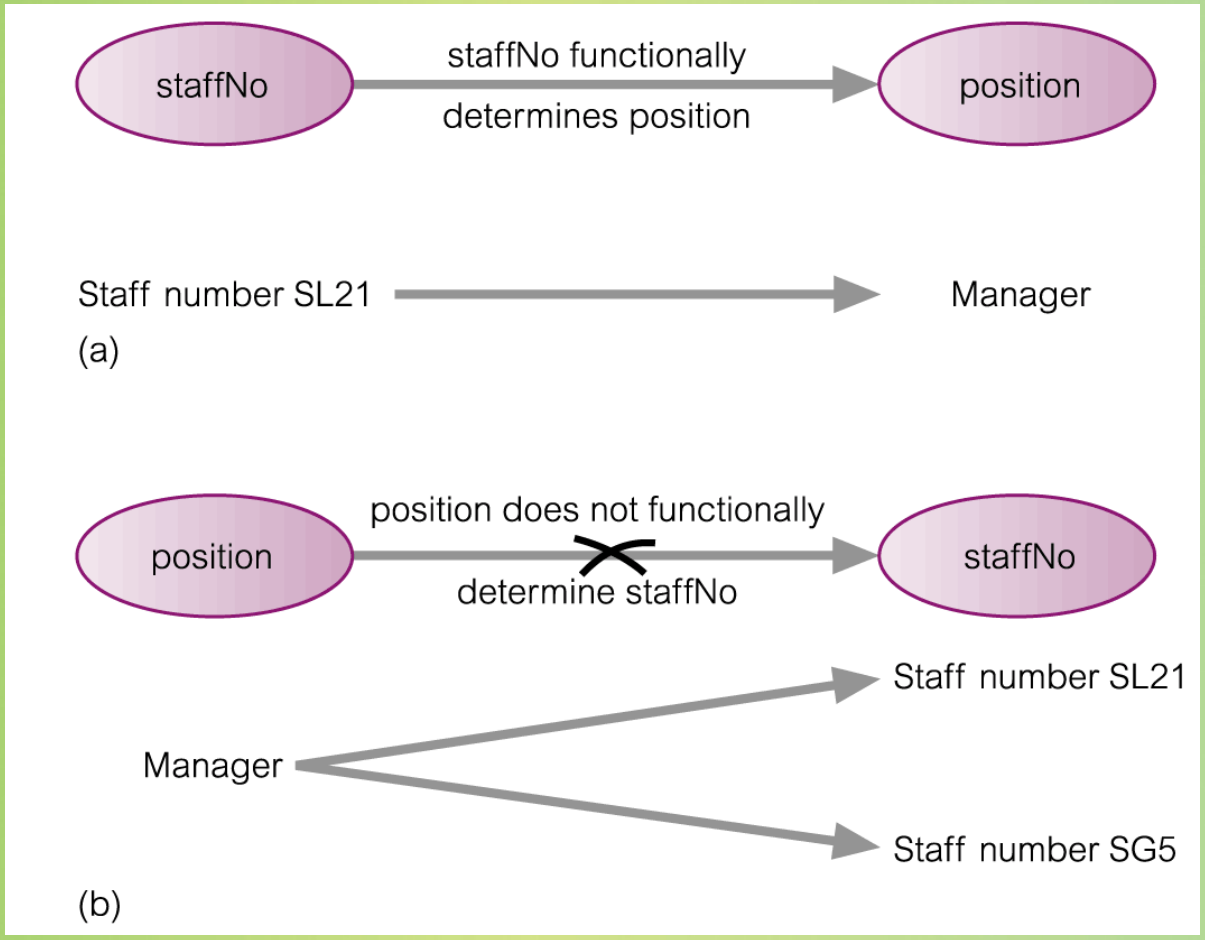
Branch

branchNo	bAddress
B005	22 Deer Rd, London
B007	16 Argyll St, Aberdeen
B003	163 Main St, Glasgow

Staff Branch

staffNo	sName	position	salary	branchNo	bAddress
SL21	John White	Manager	30000	B005	22 Deer Rd, London
SG37	Ann Beech	Assistant	12000	B003	163 Main St, Glasgow
SG14	David Ford	Supervisor	18000	B003	163 Main St, Glasgow
SA9	Mary Howe	Assistant	9000	B007	16 Argyll St, Aberdeen
SG5	Susan Brand	Manager	24000	B003	163 Main St, Glasgow
SL41	Julie Lee	Assistant	9000	B005	22 Deer Rd, London

An Example Functional Dependency



Example Functional Dependency that holds for all Time

- **Based on sample data, the following functional dependencies appear to hold.**

  staffNo → sName

  sName → staffNo

Example Functional Dependency that holds for all Time

- However, the only functional dependency that remains true for all possible values for the staffNo and sName attributes of the Staff relation is:

staffNo → sName

Characteristics of Functional Dependencies

- **Determinants should have the minimal number of attributes necessary to maintain the functional dependency with the attribute(s) on the right hand-side.**

- **This requirement is called *full functional dependency.***

Characteristics of Functional Dependencies

- **Full functional dependency** indicates that if A and B are attributes of a relation, B is fully functionally dependent on A, if B is functionally dependent on A, but not on any proper subset of A.

Sample Data

Staff

staffNo	sName	position	salary	branchNo
SL21	John White	Manager	30000	B005
SG37	Ann Beech	Assistant	12000	B003
SG14	David Ford	Supervisor	18000	B003
SA9	Mary Howe	Assistant	9000	B007
SG5	Susan Brand	Manager	24000	B003
SL41	Julie Lee	Assistant	9000	B005

Branch

branchNo	bAddress
B005	22 Deer Rd, London
B007	16 Argyll St, Aberdeen
B003	163 Main St, Glasgow

Staff Branch

staffNo	sName	position	salary	branchNo	bAddress
SL21	John White	Manager	30000	B005	22 Deer Rd, London
SG37	Ann Beech	Assistant	12000	B003	163 Main St, Glasgow
SG14	David Ford	Supervisor	18000	B003	163 Main St, Glasgow
SA9	Mary Howe	Assistant	9000	B007	16 Argyll St, Aberdeen
SG5	Susan Brand	Manager	24000	B003	163 Main St, Glasgow
SL41	Julie Lee	Assistant	9000	B005	22 Deer Rd, London

Example Full Functional Dependency

- Exists in the Staff relation

  staffNo, sName → branchNo

- True - each value of (staffNo, sName) is associated with a single value of branchNo.

- However, branchNo is also functionally dependent on a subset of (staffNo, sName), namely staffNo. Example above is a *partial dependency.*

X→ Y : for every pair t1 and t2 if t1.X = t2.X, then t1.Y = t2.Y

- α is a superkey for R iff $\alpha \rightarrow R$.
  where R is taken as the schema for relation R.
- α is a candidate key for R iff
  - $\alpha \rightarrow R$, and
  - for no γ that is a proper subset of α, $\gamma \rightarrow R$.

student_ID	student_name	course_ID	course_name
111	Chan Tai Man	3170	Database
222	Wong Siu Ling	3170	Database
333	Tam Wai Ming	3160	Algorithms
111	Chan Tai Man	3160	Algorithms

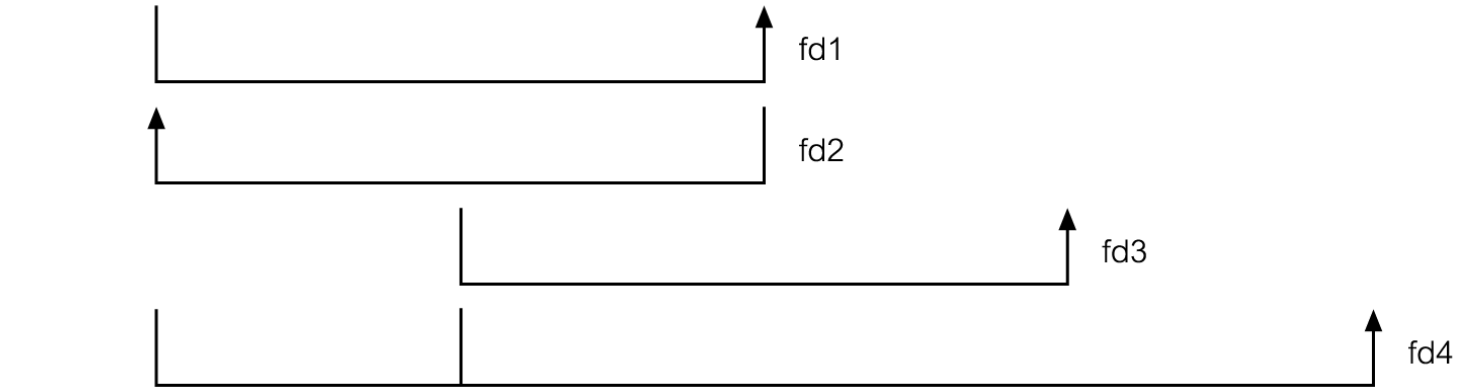
(student_ID, course_ID) is a candidate key

(student_ID, course_ID, course_name) is a superkey but not a candidate key

Example - Using sample data to identify functional dependencies.

Sample Relation

A	B	C	D	E
a	b	z	w	q
e	b	r	w	p
a	d	z	w	t
e	d	r	w	q
a	f	z	s	t
e	f	r	s	t



Example - Using sample data to identify functional dependencies.

- **Function dependencies between attributes A to E in the Sample relation.**

$A \rightarrow C$	(fd1)
$C \rightarrow A$	(fd2)
$B \rightarrow D$	(fd3)
$A, B \rightarrow E$	(fd4)

Identifying the Primary Key for a Relation using Functional Dependencies

- Main purpose of identifying a set of functional dependencies for a relation is to specify the set of integrity constraints that must hold on a relation.

- An important integrity constraint to consider first is the identification of candidate keys, one of which is selected to be the primary key for the relation.

Example - Identify Primary Key for StaffBranch Relation

- The determinants are staffNo, branchNo, bAddress, (branchNo, position), and (bAddress, position).

- To identify all candidate key(s), identify the attribute (or group of attributes) that uniquely identifies each tuple in this relation.

Example - Identifying Primary Key for StaffBranch Relation

- All attributes that are not part of a candidate key should be functionally dependent on the key.

- The only candidate key and therefore primary key for StaffBranch relation, is staffNo, as *all* other attributes of the relation are functionally dependent on staffNo.

Example - Identifying Primary Key for Sample Relation

- Sample relation has four functional dependencies
- The determinants in the Sample relation are A, B, C, and (A, B). However, the only determinant that functionally determines all the other attributes of the relation is (A, B).

- (A, B) is identified as the primary key for this relation.

- The semantics of an application is a set of constraints *imposed by the application*.

- This set of constraints are set up either explicitly or implicitly.

  Explicitly: by domain experts
  Implicitly: by 'common sense'

Example:

- Application is to keep track of information about employees in a company.

- Information to be kept track of includes:

  eid: employee's id number
  ename: employee name
  address: address of the employee
  gender: employee's gender
  dname: name of the department that the employee works for
  dhname: department head's name
  dhgender: department head's gender

Let's construct a relation schema as follows:

Employee

eid	ename	address	gender	dname	dhname	dhgender

Which of the following dependencies are true?

√ 1. eid → ename
× 2. ename → eid
√ 3. eid → dhname
√ 4. eid → dhgender
× 5. gender → address
√ 6. dname → dhname
× 7. dhname → address
× 8. dhname → dname

Given

a:Employee's id number is unique

b:Each employee works for only one dept.

c:Each department has only one head

Implicit: common sense

Reasoning about FDs

F – a set of functional dependencies

f – an individual functional dependency

f is implied by F if whenever all functional dependencies in F are true, then f is true.

Workers(id, name, office, did, since)

{ id $\rightarrow$ did, did $\rightarrow$ office }

implies:

id $\rightarrow$ office

student_ID	student_name	course_ID	course_name	department_name
111	Chan	3170	DB	CSE
222	Wong	3170	DB	CSE
333	Tam	3160	Cal	MATH
111	Chan	3160	Cal	MATH

reflexivity:

trivial dependency
LHS contains RHS

student_ID, student_name → student_ID

student_ID, student_name → student_name

augmentation:

student_ID → student_name

implies

student_ID, course_name → student_name, course_name

transitivity:

course_ID → course_name and course_name →department_name

Implies course_ID → department_name

Recall X→ Y : for every pair t1 and t2 if t1.X = t2.X, then t1.Y = t2.Y

Sample Data

Staff

staffNo	sName	position	salary	branchNo
SL21	John White	Manager	30000	B005
SG37	Ann Beech	Assistant	12000	B003
SG14	David Ford	Supervisor	18000	B003
SA9	Mary Howe	Assistant	9000	B007
SG5	Susan Brand	Manager	24000	B003
SL41	Julie Lee	Assistant	9000	B005

Branch

branchNo	bAddress
B005	22 Deer Rd, London
B007	16 Argyll St, Aberdeen
B003	163 Main St, Glasgow

Staff Branch

staffNo	sName	position	salary	branchNo	bAddress
SL21	John White	Manager	30000	B005	22 Deer Rd, London
SG37	Ann Beech	Assistant	12000	B003	163 Main St, Glasgow
SG14	David Ford	Supervisor	18000	B003	163 Main St, Glasgow
SA9	Mary Howe	Assistant	9000	B007	16 Argyll St, Aberdeen
SG5	Susan Brand	Manager	24000	B003	163 Main St, Glasgow
SL41	Julie Lee	Assistant	9000	B005	22 Deer Rd, London

Example Transitive Dependency

- Consider functional dependencies in the StaffBranch relation

  staffNo → sName, position, salary, branchNo, bAddress

  branchNo → bAddress

- Transitive dependency, branchNo → bAddress exists on staffNo via branchNo.

Closure of a set of FDs

- The set of all FDs implied by a given set F of FDs is called the closure of F, denoted as F^+.

- Armstrong's Axioms, can be applied repeatedly to infer all FDs implied by a set of FDs.

Suppose X,Y, and Z are sets of attributes over a relation.

Armstrong's Axioms

- Reflexivity: if $Y \subseteq X$, then $X \rightarrow Y$
- Augmentation: if $X \rightarrow Y$, then $XZ \rightarrow YZ$
- Transitivity: if $X \rightarrow Y$ and $Y \rightarrow Z$, then $X \rightarrow Z$

- Armstrong's Axioms is sound and complete.
  - Sound: they generate only FDs in F^+.
  - Complete: repeated application of these rules will generate all FDs in F^+.
- The proof of soundness is straight forward, but completeness is harder to prove.

Proof of soundness of Armstrong's Axioms

Reflexivity: If $Y \subseteq X$, then $X \rightarrow Y$

e.g. sid, sname $\rightarrow$ sname

Assume $\exists$ t_1, t_2 such that $t_1.X = t_2.X$

then $t_1.Y = t_2.Y$ ----- since $Y \subseteq X$

Hence $X \rightarrow Y$

Recall $X \rightarrow Y$: for every pair t1 and t2 if t1.X = t2.X, then t1.Y = t2.Y

Augmentation: if $X \rightarrow Y$, then $XZ \rightarrow YZ$

Assume $\exists\ t_1, t_2$ such that $t_1.XZ = t_2.XZ$

$t_1.Z = t_2.Z$, ------ (1)

$t_1.X = t_2.X$,

$t_1.Y = t_2.Y$ ---- definition of $X \rightarrow Y$ ------ (2)

$t_1.YZ = t_2.YZ$ ----from (1) and (2)

Hence, $XZ \rightarrow YZ$

Recall $X \rightarrow Y$: for every pair t1 and t2 if t1.X = t2.X, then t1.Y = t2.Y

Transitivity: If $X \rightarrow Y$ and $Y \rightarrow Z$, then $X \rightarrow Z$.

Assume $\exists\ t_1, t_2$ such that $t_1.X = t_2.X$

Then $t_1.Y = t_2.Y$ --- definition of $X \rightarrow Y$

Hence, $t_1.Z = t_2.Z$ --- definition of $Y \rightarrow Z$

Therefore, $X \rightarrow Z$

Recall $X \rightarrow Y$: for every pair t1 and t2 if t1.X = t2.X, then t1.Y = t2.Y

Additional rules

- Sometimes, it is convenient to use some additional rules while reasoning about F^+.

  - **Union**: if $X \rightarrow Y$ and $X \rightarrow Z$, then $X \rightarrow YZ$.

  - **Decomposition**: if $X \rightarrow YZ$, then $X \rightarrow Y$ and $X \rightarrow Z$.

- These additional rules are not essential in the sense that their soundness can be proved using Armstrong's Axioms.

To show correctness of the union rule:

$X \rightarrow Y$ and $X \rightarrow Z$, then $X \rightarrow YZ$ (**union**)

Proof:

$X \rightarrow Y$	… (1) (given)
$X \rightarrow Z$	… (2) (given)
$XX \rightarrow XY$	… (3) (augmentation on (1))
$X \rightarrow XY$	… (4) (simplify (3))
$XY \rightarrow ZY$	… (5) (augmentation on (2))
$X \rightarrow ZY$	… (6) (transitivity on (4) and (5))

correctness of the decomposition rule:

if $X \rightarrow YZ$, then $X \rightarrow Y$ and $X \rightarrow Z$ (decomposition)

Proof:

$X \rightarrow YZ$	… (1) (given)
$YZ \rightarrow Y$	… (2) (reflexivity)
$X \rightarrow Y$	… (3) (transitivity on (1), (2))
$YZ \rightarrow Z$	… (4) (reflexivity)
$X \rightarrow Z$	… (5) (transitivity on (1), (4))

(Leave it to you)

$R = (A, B, C)$

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

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

$AB \rightarrow AB,\ BC \rightarrow BC,\ AC \rightarrow AC,\ ABC \rightarrow ABC,$

$AB \rightarrow A,\ AB \rightarrow B,$

$BC \rightarrow B,\ BC \rightarrow C,$

$AC \rightarrow A,\ AC \rightarrow C,$

$ABC \rightarrow AB,\ ABC \rightarrow BC,\ ABC \rightarrow AC,$

$ABC \rightarrow A,\ ABC \rightarrow B,\ ABC \rightarrow C,$

Using reflexivity, we can generate all trivial dependencies

$A \rightarrow B,$ ... (1) (given)

$B \rightarrow C,$ ... (2) (given)

$A \rightarrow C,$ ... (3) (transitivity on (1) and (2))

$AC \rightarrow BC,$ ... (4) (augmentation on (1))

$AC \rightarrow B,$ ... (5) (decomposition on (4))

$A \rightarrow AB,$ ... (6) (augmentation on (1))

$AB \rightarrow AC,\ AB \rightarrow C,\ B \rightarrow BC,$

$A \rightarrow AC,\ AB \rightarrow BC,\ AB \rightarrow ABC,\ AC \rightarrow ABC,\ A \rightarrow BC,\ A \rightarrow ABC \}$

Attribute Closure

- Computing the closure of a set of FDs can be expensive

- In many cases, we just want to check if a given FD $X \rightarrow Y$ is in F^+.

  X - a set of attributes

  F - a set of functional dependencies

Given a set of attributes $A_1, \ldots, A_n$

The **closure**, $\{A_1, \ldots, A_n\}^+$, is the set of attributes B s.t. $A_1, \ldots, A_n \rightarrow B$

Example:

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

$A^+ = ABC$ ..... $A \rightarrow X$ where $X \subseteq ABC$ (A->A, A->B, A->C)

$B^+ = BC$

$C^+ = C$

$AB^+ = ABC$

X^+ - **closure of X under F**

set of attributes functionally determined by X under F.

Algorithm to compute closure of attributes X^+ under F

```
closure := X;
Repeat
    for each U → V in F do
    begin
        if U ⊆ closure
        then closure := closure ∪ V;
    end
Until (there is no change in closure)
```

X-> closure

closure->U (LHS contains RHS)

U->V (Imply X-> V, thus V should be put in closure

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

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

To compute AG^+

closure = AG (AG->AG)

closure = ABG ($A \rightarrow B$)

closure = $ABCG$ ($A \rightarrow C$)

closure = $ABCGH$ ($CG \rightarrow H$)

closure = $ABCGHI$ ($CG \rightarrow I$)

Is AG a candidate key?

$AG \rightarrow R$

$A^+ \rightarrow R$?

$G^+ \rightarrow R$?

To check if a given FD $AG \rightarrow HI$ is in F+,
Check if HI is in AG+

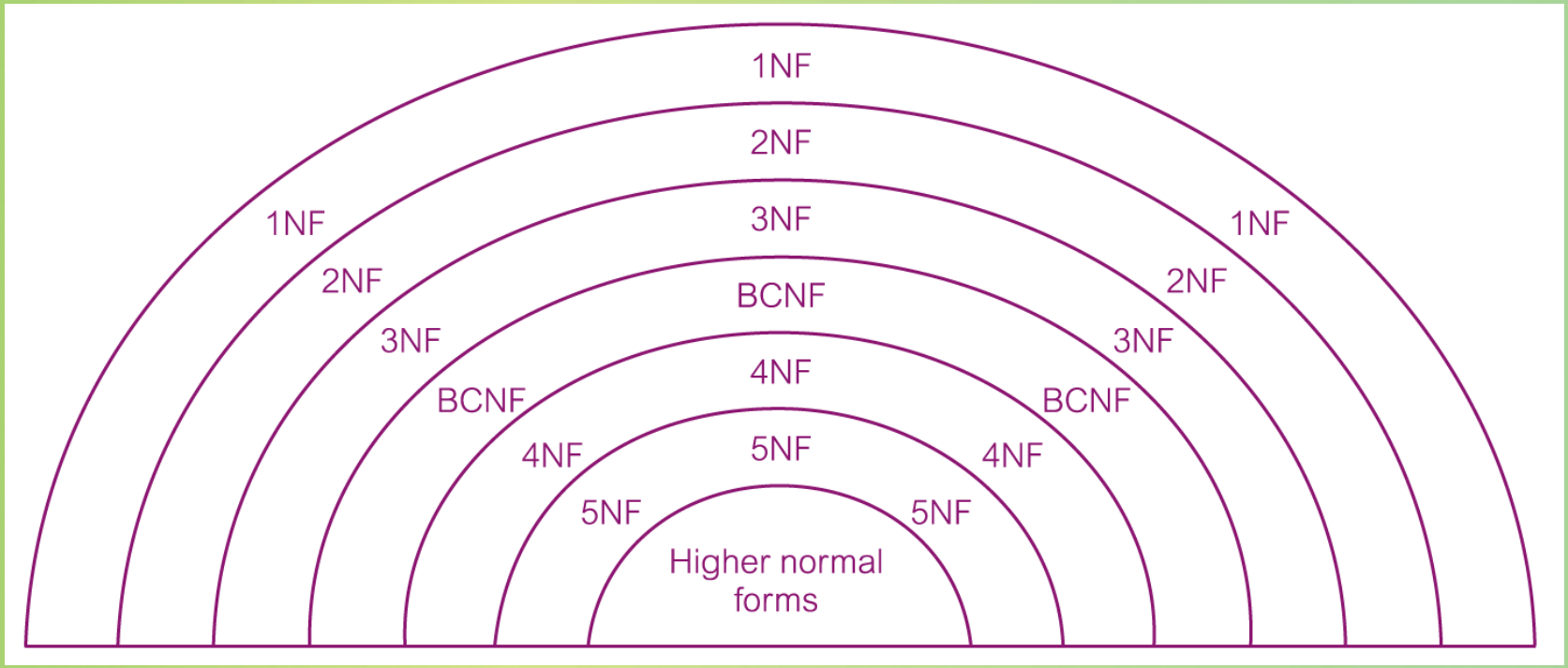
Relational Database Design

- Given a relation schema, we need to decide whether it is a good design or we need to decompose it into smaller relations.
- Such a decision must be guided by an understanding of what problems arise from the current schema.
- To provide such guidance, several normal forms have been proposed.
  - If a relation schema is in one of these normal forms, we know that certain kinds of problems cannot arise.

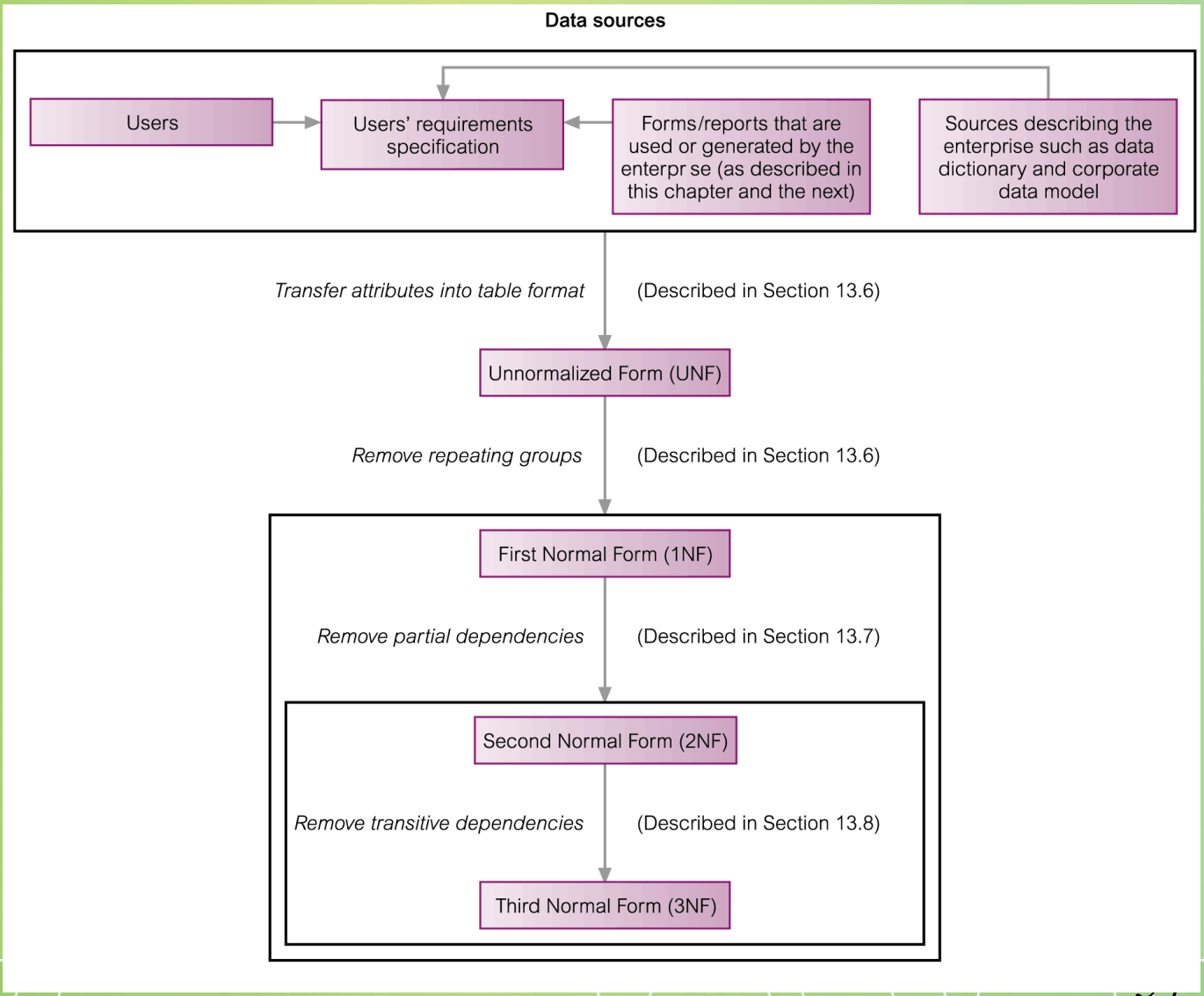
The Process of Normalization

- As normalization proceeds, the relations become progressively more restricted (stronger) in format and also less vulnerable to update anomalies.

The Process of Normalization



The Process of Normalization



Normal Forms

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

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

Unnormalized Form (UNF)

- A table that contains one or more repeating groups.

- To create an unnormalized table
  - Transform the data from the information source (e.g. form) into table format with columns and rows.

Example (UMF)

ClientRental

clientNo	cName	propertyNo	pAddress	rentStart	rentFinish	rent	ownerNo	oName
CR76	John Kay	PG4	6 Lawrence St, Glasgow	1-Jul-12	31-Aug-13	350	CO40	Tina Murphy
		PG16	5 Novar Dr, Glasgow	1-Sep-13	1-Sep-14	50	CO93	Tony Shaw
CR56	Aline Stewart	PG4	6 Lawrence St, Glasgow	1-Sep-11	10-June-12	350	CO40	Tina Murphy
		PG36	2 Manor Rd, Glasgow	10-Oct-12	1-Dec-13	375	CO93	Tony Shaw
		PG16	5 Novar Dr, Glasgow	1-Nov-14	10-Aug-15	450	CO93	Tony Shaw

Figure 14.10 ClientRental unnormalized table.

First Normal Form (1NF)

- A relation in which the intersection of each row and column contains one and only one value.

UNF to 1NF

- Nominate an attribute or group of attributes to act as the key for the unnormalized table.
- Identify the repeating group(s) in the unnormalized table which repeats for the key attribute(s).

UNF to 1NF

- Remove the repeating group by
  - Entering appropriate data into the empty columns of rows containing the repeating data ('flattening' the table).
  - Or by
  - Placing the repeating data along with a copy of the original key attribute(s) into a separate relation.

UMF to 1NF

ClientRental

clientNo	propertyNo	cName	pAddress	rentStart	rentFinish	rent	ownerNo	oName
CR76	PG4	John Kay	6 Lawrence St, Glasgow	1-Jul-12	31-Aug-13	350	CO40	Tina Murphy
CR76	PG16	John Kay	5 Novar Dr, Glasgow	1-Sep-13	1-Sep-14	450	CO93	Tony Shaw
CR56	PG4	Aline Stewart	6 Lawrence St, Glasgow	1-Sep-11	10-Jun-12	350	CO40	Tina Murphy
CR56	PG36	Aline Stewart	2 Manor Rd, Glasgow	10-Oct-12	1-Dec-13	375	CO93	Tony Shaw
CR56	PG16	Aline Stewart	5 Novar Dr, Glasgow	1-Nov-14	10-Aug-15	450	CO93	Tony Shaw

Figure 14.11 First Normal Form ClientRental relation.

Second Normal Form (2NF)

- Based on the concept of full functional dependency.

- Full functional dependency indicates that if
  - A and B are attributes of a relation,
  - B is fully dependent on A if B is functionally dependent on A but not on any proper subset of A.

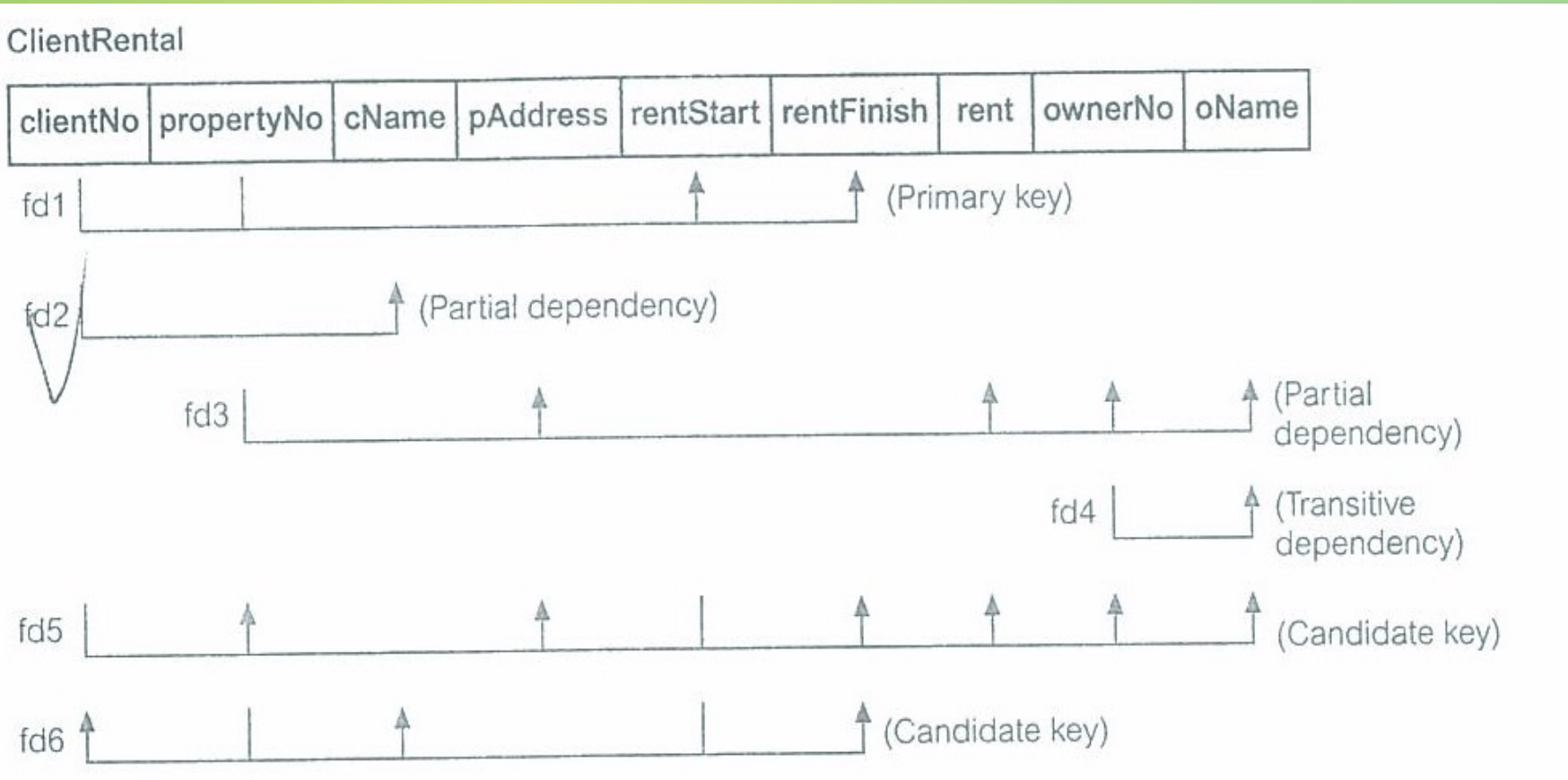
Second Normal Form (2NF)

- A relation that is in 1NF and every non-primary-key attribute is fully functionally dependent on the primary key.

1NF to 2NF

- Identify the primary key for the 1NF relation.
- Identify the functional dependencies in the relation.
- If partial dependencies exist on the primary key, remove them by creating a new relation along with a copy of their determinant.

FDs in ClientRental relation



Remove partial dependency

Client

clientNo	cName
CR76	John Kay
CR56	Aline Stewart

PropertyRentalOwner

clientNo	propertyNo	pAddress	rentStart	rentFinish	rent	ownerNo	oName
CR76	PG4	6 Lawrence St, Glasgow	1-Jul-12	31-Aug-13	350	CO40	Tina Murphy
CR76	PG16	5 Novar Dr, Glasgow	1-Sep-13	1-Sep-14	450	CO93	Tony Shaw
CR56	PG4	6 Lawrence St, Glasgow	1-Sep-11	10-Jun-12	350	CO40	Tina Murphy
CR56	PG36	2 Manor Rd, Glasgow	10-Oct-12	1-Dec-13	375	CO93	Tony Shaw
CR56	PG16	5 Novar Dr, Glasgow	1-Nov-14	10-Aug-15	450	CO93	Tony Shaw

Figure 14.13 Alternative 1NF Client and PropertyRental-Owner relations.

2NF

Client

clientNo	cName
CR76	John Kay
CR56	Aline Stewart

Rental

clientNo	propertyNo	rentStart	rentFinish
CR76	PG4	1-Jul-12	31-Aug-13
CR76	PG16	1-Sep-13	1-Sep-14
CR56	PG4	1-Sep-11	10-Jun-12
CR56	PG36	10-Oct-12	1-Dec-13
CR56	PG16	1-Nov-14	10-Aug-15

PropertyOwner

propertyNo	pAddress	rent	ownerNo	oName
PG4	6 Lawrence St, Glasgow	350	CO40	Tina Murphy
PG16	5 Novar Dr, Glasgow	450	CO93	Tony Shaw
PG36	2 Manor Rd, Glasgow	375	CO93	Tony Shaw

Figure 14.14 Second normal form relations derived from the ClientRental relation.

Third Normal Form (3NF)

- Based on the concept of transitive dependency.
- Transitive Dependency is a condition where
  - A, B and C are attributes of a relation such that if A $\rightarrow$ B and B $\rightarrow$ C,
  - then C is transitively dependent on A through B. (Provided that A is not functionally dependent on B or C).

Third Normal Form (3NF)

- A relation that is in 1NF and 2NF and in which no non-primary-key attribute is transitively dependent on the primary key.

2NF to 3NF

- Identify the primary key in the 2NF relation.
- Identify functional dependencies in the relation.
- If transitive dependencies exist on the primary key remove them by placing them in a new relation along with a copy of their dominant.

2NF -> 3NF

Client

fd2	clientNo ® cName	(Primary key)

Rental

fd1	clientNo, propertyNo ® rentStart, rentFinish	(Primary key)
fd5′	clientNo, rentStart ® propertyNo, rentFinish	(Candidate key)
fd6′	propertyNo, rentStart ® clientNo, rentFinish	(Candidate key)

PropertyOwner

fd3	propertyNo ® pAddress, rent, ownerNo, oName	(Primary key)
fd4	ownerNo ® oName	(Transitive dependency)

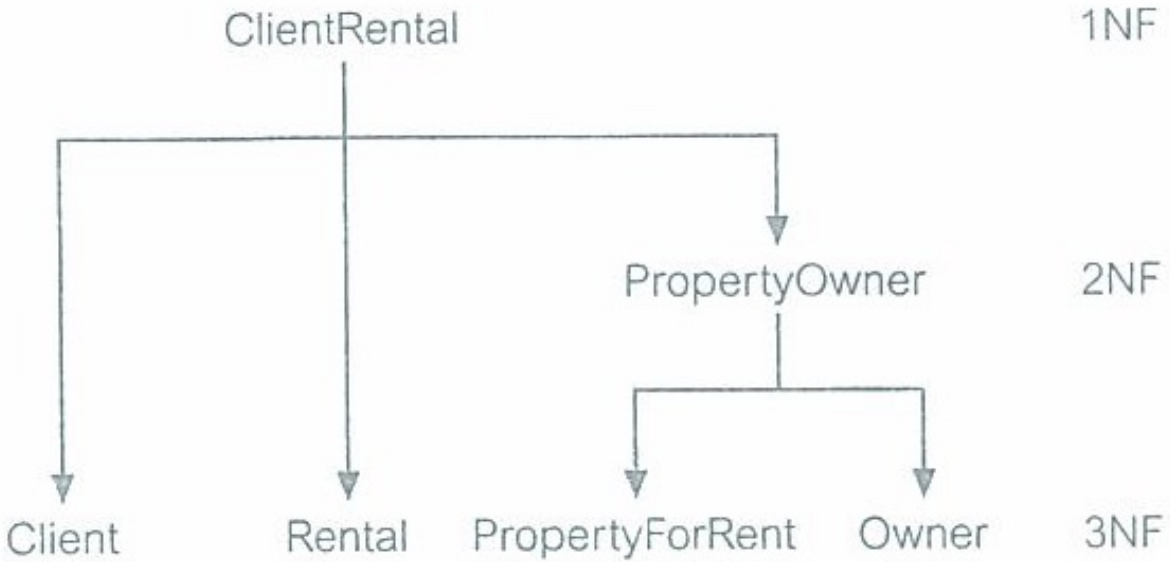
PropertyForRent

propertyNo	pAddress	rent	ownerNo
PG4	6 Lawrence St, Glasgow	350	CO40
PG16	5 Novar Dr, Glasgow	450	CO93
PG36	2 Manor Rd, Glasgow	375	CO93

Owner

ownerNo	oName
CO40	Tina Murphy
CO93	Tony Shaw

The decomposition



General Definitions of 2NF and 3NF

- **Second normal form (2NF)**
  - **A relation that is in first normal form and every non-primary-key attribute is fully functionally dependent on *any candidate key*.**
- **Third normal form (3NF)**
  - **A relation that is in first and second normal form and in which no non-primary-key attribute is transitively dependent on *any candidate key*.**

• Boyce-Codd Normal Form (BCNF)

Role of FDs in detecting redundancy:

- consider a relation R with three attributes, A,B,C
  If no non-trivial FDs hold, no potential redundancy

  If $A \rightarrow B$, then tuples with the same A value will have the same (redundant) B values.

  But if A is a superkey, then each A value is unique

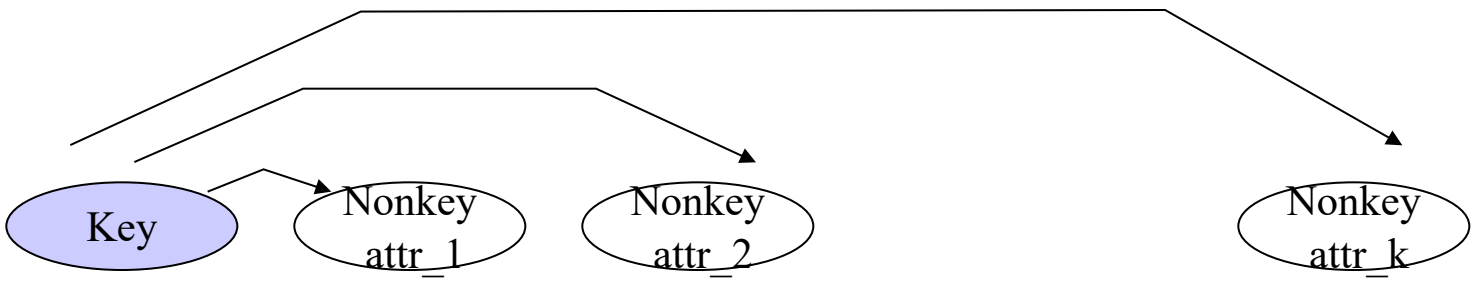
R - a relation schema, X - set of attributes, A - attribute
F - set of functional dependencies that holds over R

R is in BCNF if for any $X \rightarrow A$ in F,

- $X \rightarrow A$ is a trivial functional dependency, i.e., $A \subseteq X$.
  OR
- X is a superkey for R.

- Intuitively, in a BCNF relation, the only nontrivial dependencies are those in which a superkey determines some attributes.

- Each tuple can be thought of as an entity or relationship, identified by a key and described by the remaining attributes



FDs in a BCNF Relation

R is in BCNF if for any $X \rightarrow A$ in F,
$X \rightarrow A$ is a trivial functional dependency, i.e., $A \subseteq X$.
OR
X is a superkey for R.

Example 1:

R = (A, B, C)

F = { A → B, B → C }

Key = { A }

R is not in BCNF

A	B	C
a1	b1	c1
a2	b1	c1
a3	b1	c1
a4	b2	c2

- Decomposition into

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

- R_1 and R_2 are in BCNF

A	B
a1	b1
a2	b1
a3	b1
a4	b2

A → B

B	C
b1	c1
b2	c2

B → C

R is in BCNF if for any $X \rightarrow A$ in F,
$X \rightarrow A$ is a trivial functional dependency, i.e., $A \subseteq X$.
OR
X is a superkey for R.

Example 2:

R = (A, B, C)

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

Key = { AC }

R is not in BCNF

A	B	C
a1	b1	c1
a2	b1	c1
a3	b1	c1
a4	b2	c2

- Decomposition into

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

- R_1 and R_2 are in BCNF

A	B
a1	b1
a2	b1
a3	b1
a4	b2

$A \rightarrow B$

B	C
b1	c1
b2	c2

$C \rightarrow B$

Boyce–Codd Normal Form (BCNF)

- Based on functional dependencies that take into account all candidate keys in a relation, however BCNF also has additional constraints compared with the general definition of 3NF.

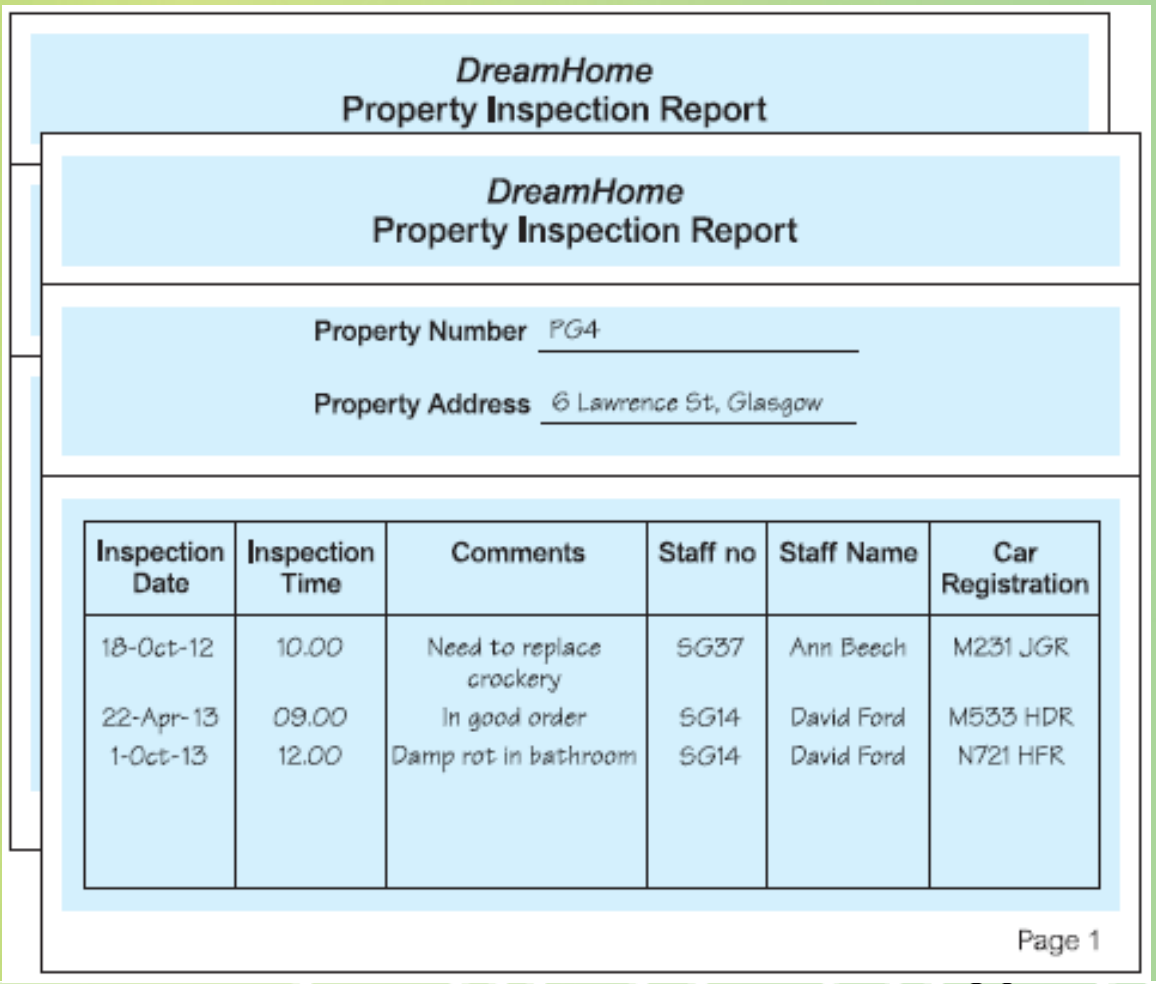
- Boyce–Codd normal form (BCNF)
  - A relation is in BCNF if and only if every determinant is a candidate key.

Boyce–Codd Normal Form (BCNF)

- Based on functional dependencies that take into account all candidate keys in a relation, however BCNF also has additional constraints compared with the general definition of 3NF.

- Boyce–Codd normal form (BCNF)
  - A relation is in BCNF if and only if every determinant is a candidate key.

Review of Normalization (UNF to BCNF)



DreamHome
Property Inspection Report

DreamHome
Property Inspection Report

Property Number PG4

Property Address 6 Lawrence St, Glasgow

Inspection Date	Inspection Time	Comments	Staff no	Staff Name	Car Registration
18-Oct-12	10.00	Need to replace crockery	SG37	Ann Beech	M231 JGR
22-Apr-13	09.00	In good order	SG14	David Ford	M533 HDR
1-Oct-13	12.00	Damp rot in bathroom	SG14	David Ford	N721 HFR

Page 1

Review of Normalization (UNF to BCNF)

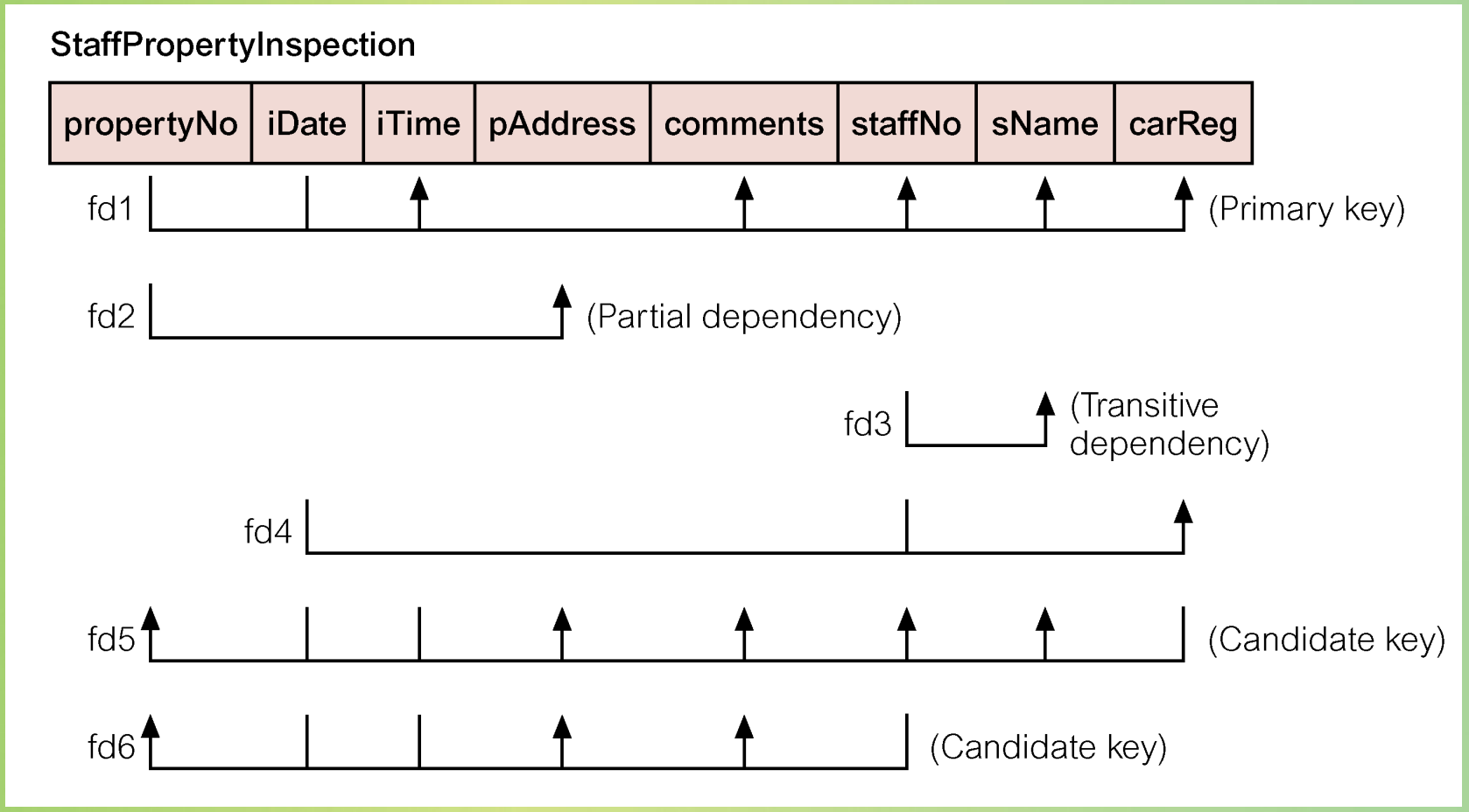
StaffPropertyInspection

propertyNo	pAddress	iDate	iTime	comments	staffNo	sName	carReg
PG4	6 Lawrence St, Glasgow	18-Oct-12 22-Apr-13 1-Oct-13	10.00 09.00 12.00	Need to replace crockery In good order Damp rot in bathroom	SG37 SG14 SG14	Ann Beech David Ford David Ford	M231 JGR M533 HDR N721 HFR
PG16	5 Novar Dr, Glasgow	22-Apr-13 24-Oct-13	13.00 14.00	Replace living room carpet Good condition	SG14 SG37	David Ford Ann Beech	M533 HDR N721 HFR

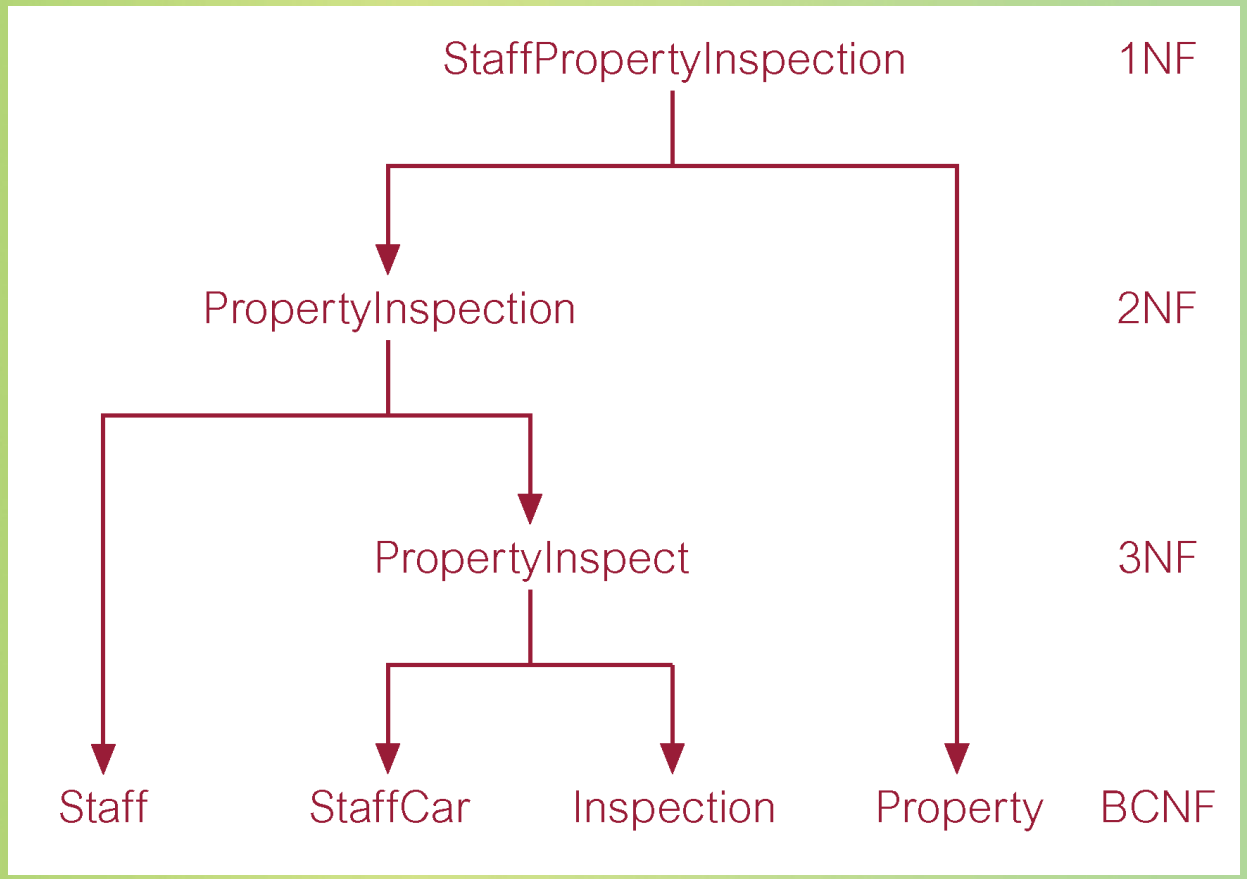
StaffPropertyInspection

propertyNo	iDate	iTime	pAddress	comments	staffNo	sName	carReg
PG4	18-Oct-12	10.00	6 Lawrence St, Glasgow	Need to replace crockery	SG37	Ann Beech	M231 JGR
PG4	22-Apr-13	09.00	6 Lawrence St, Glasgow	In good order	SG14	David Ford	M533 HDR
PG4	1-Oct-13	12.00	6 Lawrence St, Glasgow	Damp rot in bathroom	SG14	David Ford	N721 HFR
PG16	22-Apr-13	13.00	5 Novar Dr, Glasgow	Replace living room carpet	SG14	David Ford	M533 HDR
PG16	24-Oct-13	14.00	5 Novar Dr, Glasgow	Good condition	SG37	Ann Beech	N721 HFR

Review of Normalization (UNF to BCNF)



Review of Normalization (UNF to BCNF)



Third Normal Form

A relation schema R is in 3NF if,

for all $X \rightarrow A$ (A is an attribute that holds over R

- $A \in X$ (i.e., $X \rightarrow A$ is a trivial FD), or
- X is a superkey, or
- A is part of some key for R

- The definition of 3NF is similar to that of BCNF, with the only difference being the third condition.

  If R is in BCNF, obviously it is in 3NF.

- A key for a relation is a minimal set of attributes that uniquely determines all other attributes.
  - A must be part of a key (any key, if there are several).

Motivation of 3NF

- By making an exception for certain dependencies involving key attributes,

  we can ensure that every relation schema can be decomposed into a collection of 3NF relations using only lossless-join, dependency-preserving decompositions.

- Such a guarantee does not exist for BCNF relations.

- It weakens the BCNF requirements just enough to make this guarantee possible.

- Unlike BCNF, some redundancy is possible with 3NF.
  - The problems associate with partial and transitive dependencies persist if there is a nontrivial dependency

    $X \rightarrow A$ and

    X is not a superkey,

    even if the relation is in 3NF because

    A is part of a key.

Reserves

sid	bid	day	cardno
1	b1	April 5, 00	2323
2	b2	May 10, 01	1000
1	b3	January 5, 01	2323
3	b4	June 1, 00	4000

- Assume: sid $\rightarrow$ cardno (a sailor uses a unique credit card to pay for reservations).
- Reserves is not in 3NF
  - (sid, bid, day) is the only key.
  - sid $\rightarrow$ cardno violates 3NF because
    - sid is not a key and cardno is not part of a key
  - (sid, cardno) pairs are redundantly stored.

Reserves

sid	bid	day	cardno
1	b1	April 5, 00	2323
2	b2	May 10, 01	1000
1	b3	January 5, 01	2323
3	b4	June 1, 00	4000

- Assume: sid $\rightarrow$ cardno, and cardno $\rightarrow$ sid (we know that credit cards also uniquely identify the owner).
- Reserves is in 3NF
  - (sid, bid, day) is a key for Reserves.
  - (cardno, bid, day) is also a key for Reserves.
  - cardno $\rightarrow$ sid does not violate 3NF, since sid is in a key
  - sid $\rightarrow$ cardno does not violate 3NF, since cardno is in a key

(sid, cardno) pairs are redundantly stored.

Decomposition

- Decomposition is a tool that allows us to eliminate redundancy.
- It is important to check that a decomposition does not introduce new problems.
  - A decomposition allows us to recover the original relation?
  - Can we check integrity constraints efficiently?

Supply

sid	status	city	part_id	qty

Supplier

sid	status	city

and

SP

sid	part_id	qty

Supplier $\cup$ SP = Supply

{ Supplier, SP } is a decomposition of Supply

A set of relation schemas { R_1, R_2, …, R_n }, with $n \geq 2$ is a **decomposition** of R if

$R_1 \cup R_2 \cup \ldots \cup R_n = R$

- Decomposition may turn non-normal form into normal form.

Suppose R is not in BCNF, and $X \rightarrow A$ is a FD where

$X \cap A = \phi$ that violates the condition.

1. Remove A from R
2. Create a new relational schema XA
3. Repeat this process until all the relations are in BCNF

Problems with decomposition

1. Some queries become more expensive.

2. Given instances of the decomposed relations, we may not be able to reconstruct the corresponding instance of the original relation – information loss.

3. Checking some dependencies may require joining the instances of the decomposed relations.

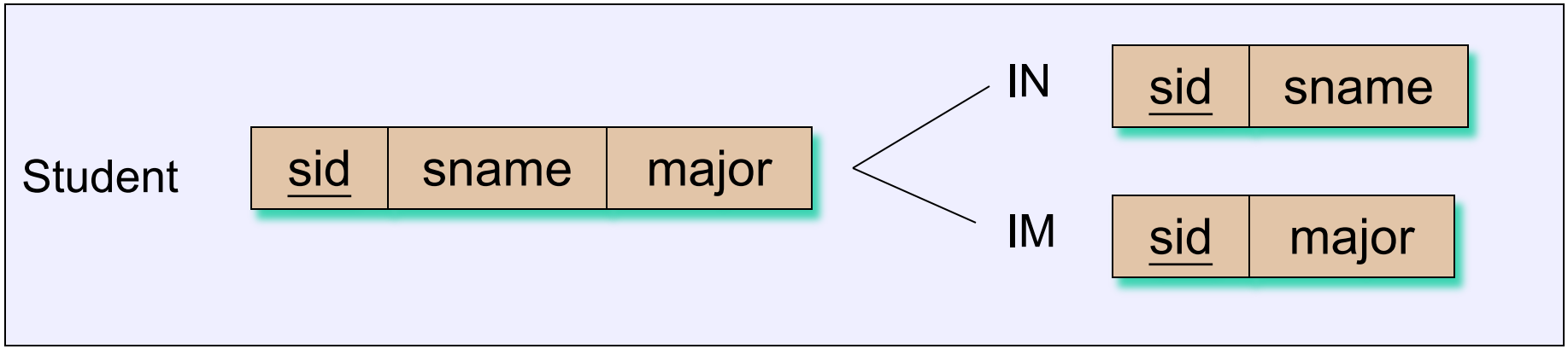
Lossless Join Decomposition

The relation schemas { R_1, R_2, …, R_n } is a **lossless-join decomposition** of R if:

for all possible relations *r* on schema *R*,

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

Example: a lossless join decomposition



Student

sid	sname	major
123	Ling Wang	C.S.
456	Hong Zhou	C.S.

IN

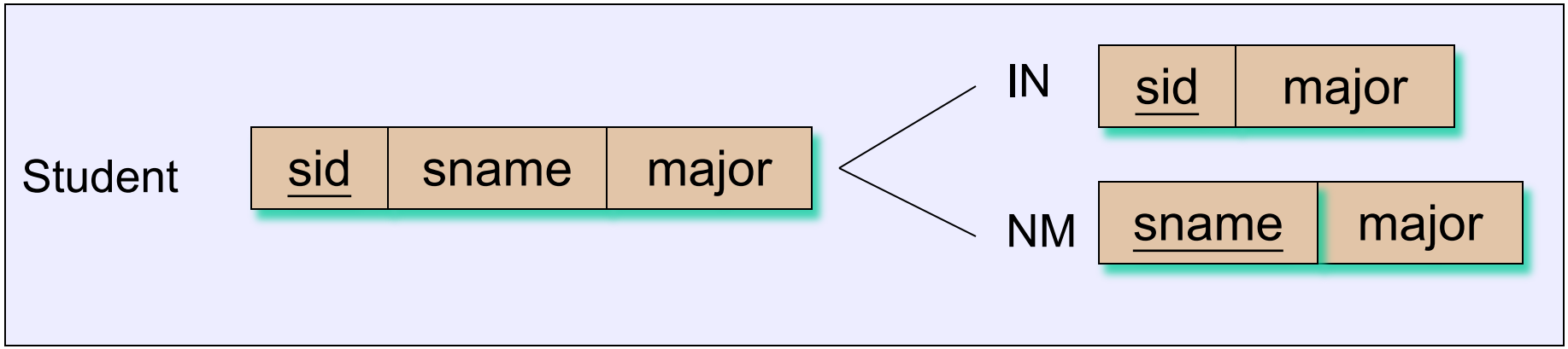
sid	sname
123	Ling Wang
456	Hong Zhou

'Student' can be recovered by joining the instances of IN and IM

IM

sid	major
123	C.S.
456	C.S.

Example: a non-lossless join decomposition



Student

sid	sname	major
123	Ling Wang	C.S.
456	Hong Zhou	C.S.

IN

sid	major
123	C.S.
456	C.S.

NM

sname	major
Ling Wang	C.S.
Hong Zhou	C.S.

Student = IN × NM????

IN

sid	major
123	C.S.
456	C.S.

NM

sname	major
Ling Wang	C.S.
Hong Zhou	C.S.

$IN \times NM$

sid	sname	major
123	Ling Wang	C.S.
123	Hong Zhou	C.S.
456	Ling Wang	C.S.
456	Hong Zhou	C.S.

$\neq$

Student

sid	sname	major
123	Ling Wang	C.S.
456	Hong Zhou	C.S.

The instance of 'Student' cannot be recovered by joining the instances of IM and NM. Therefore, such a decomposition is not a lossless join decomposition.

Theorem:

R - a relation schema

F - set of functional dependencies on R

The decomposition of R into relations with attribute sets R_1, R_2 is a lossless-join decomposition iff

$$(R_1 \cap R_2) \rightarrow R_1 \in F^+$$

OR

$$(R_1 \cap R_2) \rightarrow R_2 \in F^+$$

i.e., $R_1 \cap R_2$ is a superkey for R_1 or R_2.

(the attributes common to R_1 and R_2 must contain a key for either R_1 or R_2).

- Example
- R = (A, B, C)
- F = { $A \rightarrow B$ }

- { A, B } + { A, C } is a lossless join decomposition
  $\{A,B\} \cap \{A,C\} = \{A\}$ is a key in $\{A,B\}$

- { A, B } + { B, C } is not a lossless join decomposition
  $\{A,B\} \cap \{B,C\} = \{B\}$ is neither a key in $\{A,B\}$ nor $\{B,C\}$

- Also, consider the previous relation 'Student'

- Example
- Student = (sid, sname, major)
- F = { sid $\rightarrow$ sname, sid $\rightarrow$ major}

{ sid, sname } + { sid, major } is a lossless join decomposition

the intersection = {sid} is a key in both schemas

{sid, major} + { sname, major } is not a lossless join decomposition

the intersection = {major} is not a key in either {sid, major} or { sname, major }

Another Example

$R = \{ A, B, C, D \}$
$F = \{ A \rightarrow B, C \rightarrow D \}.$

Decomposition result: { (*A*, *B*), (*C*, *D*), (*A*, *C*) }

Consider it a two step decomposition:

1. Decompose *R* into $R_1 = (A, B)$, $R_2 = (A, C, D)$
2. Decompose R_2 into $R_3 = (C, D)$, $R_4 = (A, C)$

This is a lossless join decomposition (A, B), (C, D), (A, C)

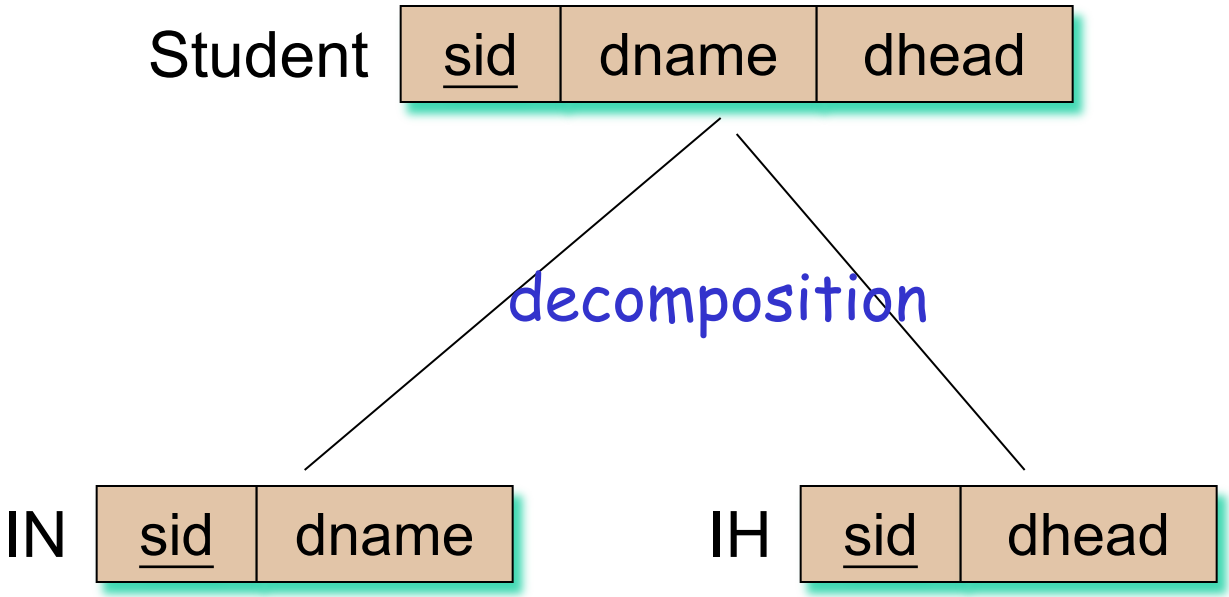
If *R* is decomposed into (*A*, *B*), (*C*, *D*)

This is a lossy-join decomposition.

Dependency Preservation In Decomposition

Dependency Preservation

Dependency set: F = { sid → dname, dname → dhead }



Dependency set: F = { sid → dname, dname → dhead }

F_{IN} = { trivial dependencies (e.g., sid, dname->sid or sid->sid),

sid → dname,
sid → sid dname}

F_{IH} = { trivial dependencies, sid → dhead, by transitivity
sid → sid dhead }

(dname → dhead) ∈ F^+ **but**

(dname → dhead) ∉ (F_{IN} U F_{IH})$^+$

This decomposition does not preserve dependency

Student

sid	dname	dhead
123	C.S.	Ying Wang
456	C.S.	Ying Wang

IN

sid	dname
123	C.S.
456	C.S.

and

IH

sid	dhead
123	Ying Wang
456	Ying Wang

Updated to

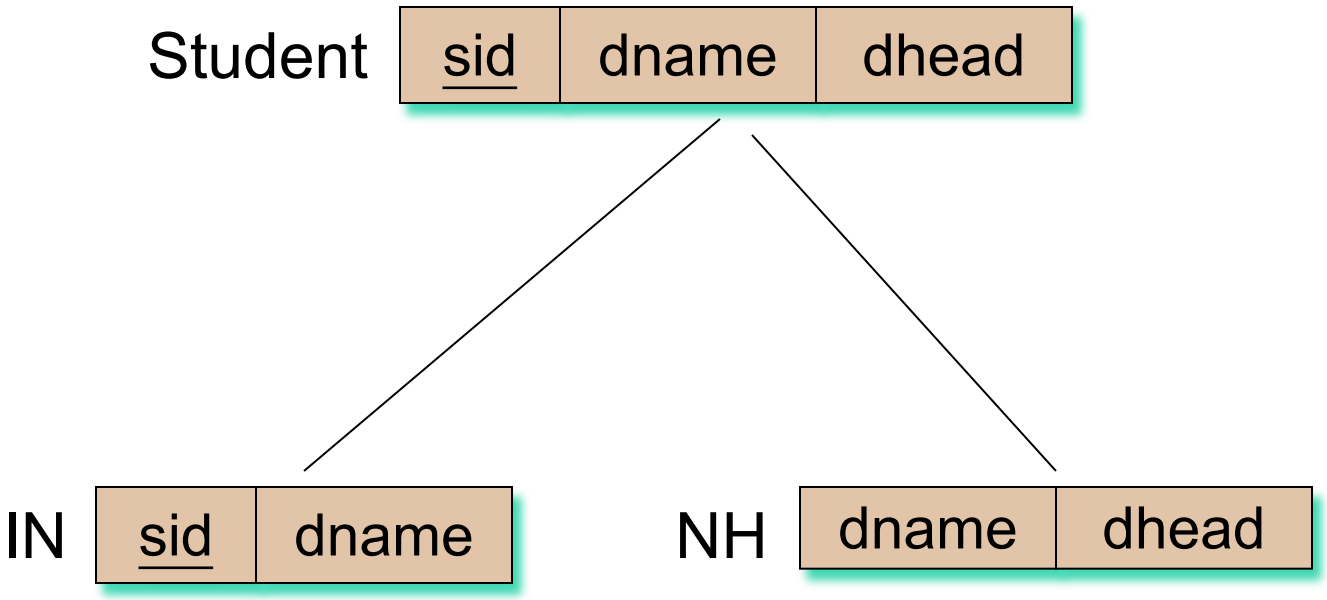
sid	dhead
123	Ying Wang
456	Lin Cheung

The update violates the FD 'dname → dhead'.

However, it can only be caught when we join IN and IH.

Dependency set: F = { sid → dname, dname → dhead }

Let's decompose the relation in another way.



Dependency set: F = { sid → dname, dname → dhead }

F_{IN} = { trivial dependencies, sid → dname,
sid → sid dname}

F_{NH} = { trivial dependencies, dname → dhead,
dname → dname dhead }

$(F_{IN} \cup F_{NH})^+ = F^+$

This decomposition preserves dependency:

Student

sid	dname	dhead
123	C.S.	Ying Wang
456	C.S.	Ying Wang

IN

sid	dname
123	C.S.
456	C.S.

and

NH

dname	dhead
C.S.	Ying Wang
C.S.	Ying Wang

Updated to

dname	dhead
C.S.	Ying Wang
C.S.	Lin Cheung

The error in NH will immediately be caught by the DBMS,

since it violates F.D. dname → dhead.

No join is necessary for the detection

(This will not be happened)

Dependency Preservation

R - a relation schema

F - set of functional dependencies on R

$\{ R_1, R_2 \}$ - a decomposition of R.

F_i - the set of dependencies in F^+ involving only attributes in R_i.

F_i is called the **projection** of F on the set of attributes of R_i.

dependency is preserved if

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

- a dependency-preserving decomposition allows us to enforce all FDs by examining a single relation instance on each insertion or modification of a tuple.

Normalization

- Converting relations to BCNF or 3NF.

- If a relation schema is not in BCNF
  - It is possible to obtain a lossless-join decomposition into a collection of BCNF relation schemas.
  - **Dependency-preserving is not guaranteed.**

- 3NF
  - There is always a dependency-preserving, lossless-join decomposition into a collection of 3NF relation schemas.

BCNF Decomposition

Suppose R is not in BCNF, A is an attribute, and $X \rightarrow A$ is a FD that violates the BCNF condition.

1. Remove A from R
2. Decompose R into XA and $R-A$
3. Repeat this process until all the relations become BCNF

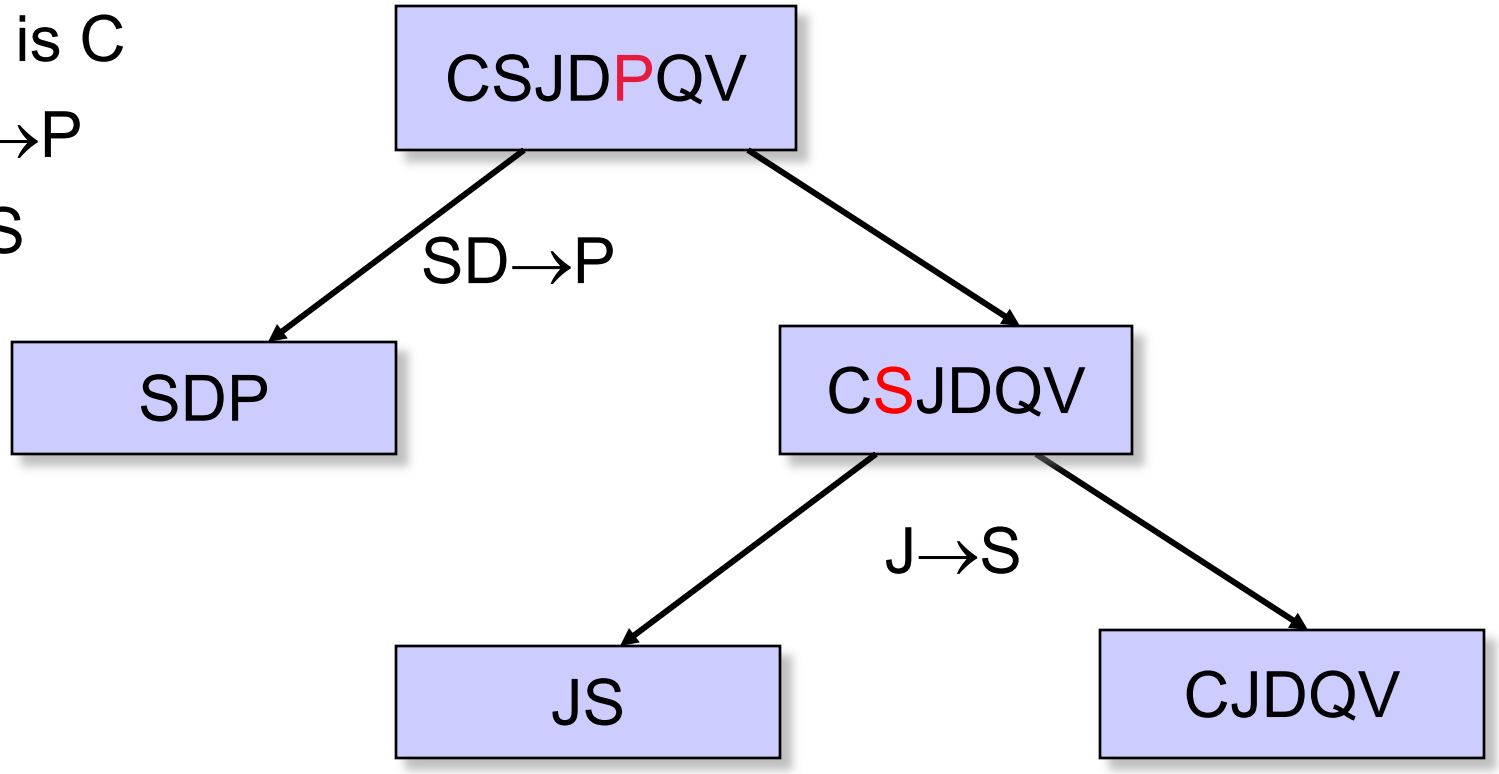
- It is a lossless join decomposition.
- But not necessary dependency preserving

Contracts(Contractid, Supplierid, proJectid, Deptid, Partid, Qty, Value)

Key is C

SD→P

J→S

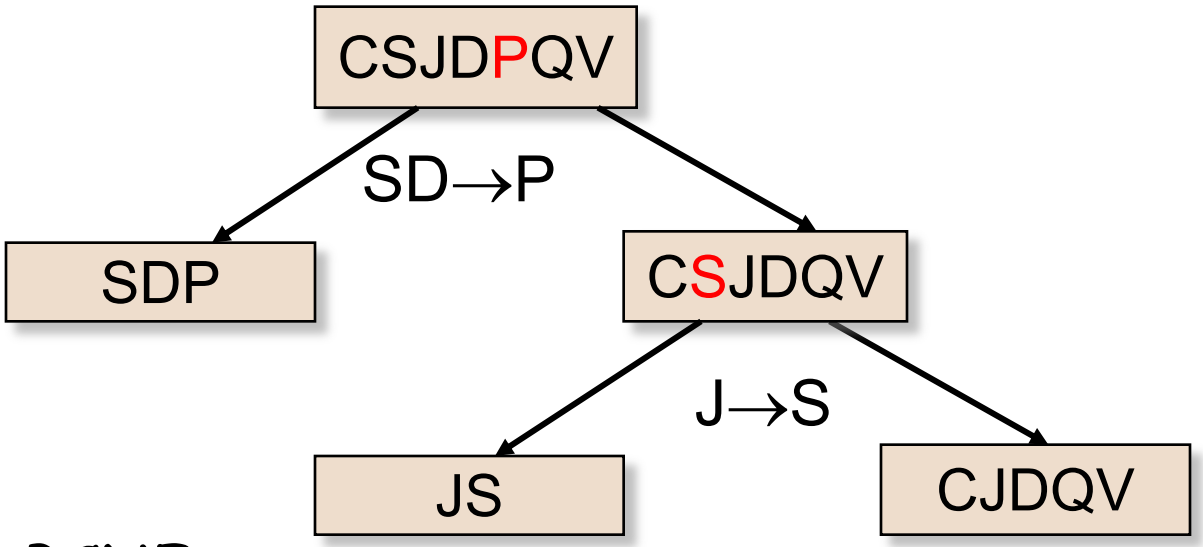


Key is C

SD→P

J→S

JP →C



The result is in BCNF

Does not preserve JP→C, we can add a schema: CJP

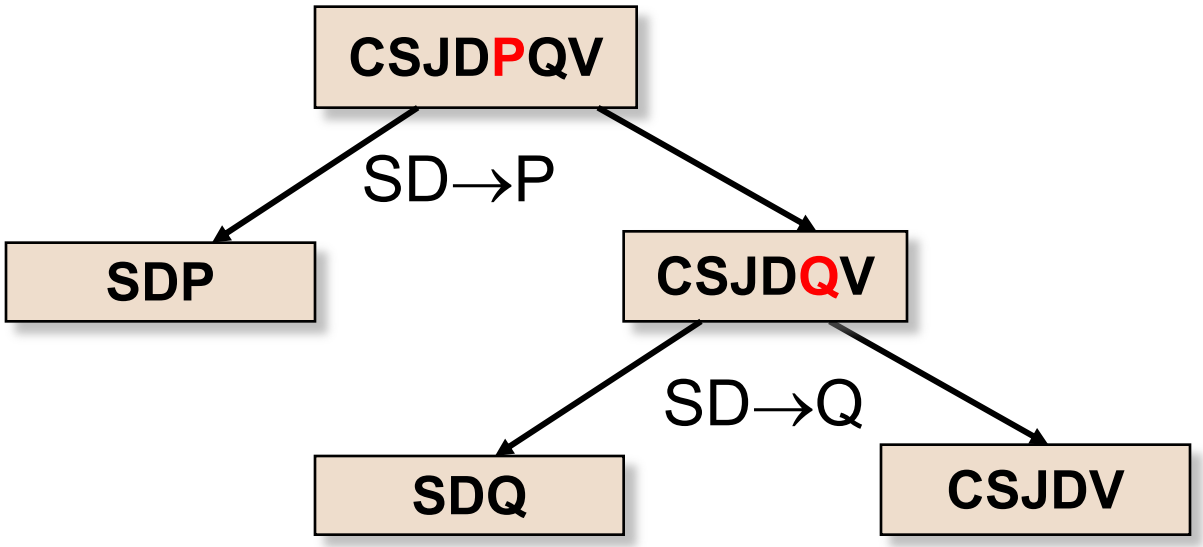
Each of SDP, JS, CJDQV, CJP is in BCNF, but there is redundancy in CJP. Why ?

$$CJP = \Pi_{c,j,p}(SDP \bowtie CJDQV)$$

Key is C

SD→P

SD→Q



SD is a key in SDP and SDQ,
There is no dependency between P and Q

we can combine SDP and SDQ into one schema
Resulting in SDPQ, CSJDV

Example

- R = (J, K, L)
  F = (JK $\rightarrow$ L, L $\rightarrow$ K)
  Two candidate keys *JK* and *JL*.
- R is not in BCNF

Any decomposition of R will fail to preserve $JK \rightarrow L$.

However, it is possible for 3NF decomposition to be both

lossless join and decomposition preserving. (Since K is part of some keys, R is 3NF)

Canonical Cover

A minimal and equivalent set of functional dependency

Two sets of functional dependencies E and F are **equivalent** if $E^+ = F^+$

Example: R = (A, B, C)

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

F can be simplified : By the decomposition rule,

$A \rightarrow BC$ implies $A \rightarrow B$ and $A \rightarrow C$

Therefore $A \rightarrow B$ is redundant.

F ' = { $A \rightarrow BC$, $B \rightarrow C$, $AB \rightarrow C$ }

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

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

- Another way to show that $A \rightarrow B$ is redundant:

  From $A \rightarrow BC, B \rightarrow C, AB \rightarrow C$,

  Compute the closure of A:

  result = A

  result = ABC, Hence $A^+ = ABC$

  Therefore $A \rightarrow B$ is redundant.

  $F' = \{ A \rightarrow BC, B \rightarrow C, AB \rightarrow C \}$

note that $A \rightarrow B$ is missing

note that this implies $A \rightarrow B$

Example (cont)

F' can be further simplified

- $F' = \{ A \rightarrow BC, B \rightarrow C, AB \rightarrow C \}$

$B \rightarrow C$ (given)

$AB \rightarrow AC$ (augmentation)

$AB \rightarrow C$ (decomposition if $X \rightarrow YZ$, then $X \rightarrow Y$ and $X \rightarrow Z$)

$AB \rightarrow C$ is redundant,

or A **is extraneous** in $AB \rightarrow C$.

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

Example (cont.)

- $F' = \{ A \rightarrow BC, B \rightarrow C, AB \rightarrow C \}$

Another way to show that A is extraneous in $AB \rightarrow C$

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

we can compute $(AB)^+$ under F'' as follows

result = AB

result = ABC (from $B \rightarrow C$)

Hence $(AB)^+ = ABC$

$AB \rightarrow C$ is redundant in F',

or A is extraneous in $AB \rightarrow C$.

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

Example (cont.)

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

C is extraneous in $A \rightarrow BC$:

note that $A \rightarrow BC$ is missing

From $A \rightarrow B$ and $B \rightarrow C$

we can deduce $A \rightarrow C$ (transitivity).

From $A \rightarrow B$ and $A \rightarrow C$

we get $A \rightarrow BC$ (union)

F‴ = { $A \rightarrow B$, $B \rightarrow C$ } .....This is a canonical cover for F

Example (cont.)

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

Another way to show that C **is extraneous** in $A \rightarrow BC$:

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

we can compute A^+ under F''' as follows

result = A

result = AB ($A \rightarrow B$)

result = ABC ($B \rightarrow C$)

Hence $A^+ = ABC$

$A \rightarrow BC$ can be deduced

$F''' = \{ A \rightarrow B, B \rightarrow C \}$ .. This is a canonical cover for F

A canonical cover F_c of a set of functional dependency F must have the following properties.

1. Every functional dependency $\alpha \rightarrow \beta$ in F_c contains no **extraneous** attributes in α

   (one that can be removed from α without changing F_c^+).

   So *A* is extraneous in α if $A \in \alpha$ and

   $$(F_c - \{\alpha \rightarrow \beta\}) \cup \{(\alpha - A) \rightarrow \beta\}$$

   logically implies F_c.

2. Every functional dependency $\alpha \rightarrow \beta$ in F_c contains no **extraneous** attributes in β (ones that can be removed from β without changing F_c^+). So A is extraneous in β if $A \in \beta$ and
$(F_c - \{\alpha \rightarrow \beta\}) \cup \{\alpha \rightarrow (\beta - A)\}$
logically implies F_c.

3. Each left side of a functional dependency in F_c is unique. That is there are no two dependencies $\alpha_1 \rightarrow \beta_1$ and $\alpha_2 \rightarrow \beta_2$ in F_c such that $\alpha_1 = \alpha_2$.

Compute a canonical cover for F:

repeat

Replace any $\alpha_1 \rightarrow \beta_1$ and $\alpha_1 \rightarrow \beta_2$
by $\alpha_1 \rightarrow \beta_1 \beta_2$

Delete any **extraneous attribute**
from any $\alpha \rightarrow \beta$

until F does not change

Example: Given $F = \{ A \rightarrow BC, A \rightarrow B, B \rightarrow AC, C \rightarrow A \}$

Combine $A \rightarrow BC$, $A \rightarrow B$ into $A \rightarrow BC$

$F' = \{ A \rightarrow BC, B \rightarrow AC, C \rightarrow A \}$

$F'' = \{ A \rightarrow B, \quad B \rightarrow AC, C \rightarrow A \}$

C is extraneous in $A \rightarrow BC$ because (You may also check B)

we can compute A^+ under F'' as follows

result = A

result = AB ($A \rightarrow B$)

result = ABC ($B \rightarrow AC$)

Hence $A^+ = ABC$

And we can deduce $A \rightarrow BC$,

Example (cont):

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

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

A is extraneous in $B \rightarrow AC$ because

we can compute B^+ under F''' as follows

result = B

result = BC ($B \rightarrow C$)

result = ABC ($C \rightarrow A$)

Hence $B^+ = ABC$

And we can deduce $B \rightarrow AC$,

$F''' = \{ A \rightarrow B,\ B \rightarrow C,\ C \rightarrow A \}$ ..... Canonical cover for F

3NF Synthesis Algorithm

Find a canonical cover F_c for F;

result = ϕ;

for each $\alpha \rightarrow \beta$ in F_c **do**

  if no schema in result contains $\alpha\beta$

   then add schema $\alpha\beta$ to result;

if no schema in result contains a candidate key for R

  then begin

    choose any candidate key α for R;

    add schema α to the result

  end

Note: result is lossless-join and dependency preserving

Example

R = (student_id, student_name, course_id, course_name)

F = { student_id → student_name,
course_id → course_name }

{ student_id, course_id } is a candidate key.

F_c = F

R_1 = (student_id, student_name)

R_2 = (course_id, course_name)

R_3 = (student_id, course_id)

Example 2

$R = (A, B, C)$

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

Key={A, B}

R is not in 3NF

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

Decomposition into: R1 = (A, B), R2 = (B, C)

R1 and R2 are in 3NF

BCNF VS 3NF

- always possible to decompose a relation into relations in 3NF and
  - the decomposition is lossless
  - dependencies are preserved

- always possible to decompose a relation into relations in BCNF and
  - the decomposition is lossless
  - may not be possible to preserve dependencies

More Examples

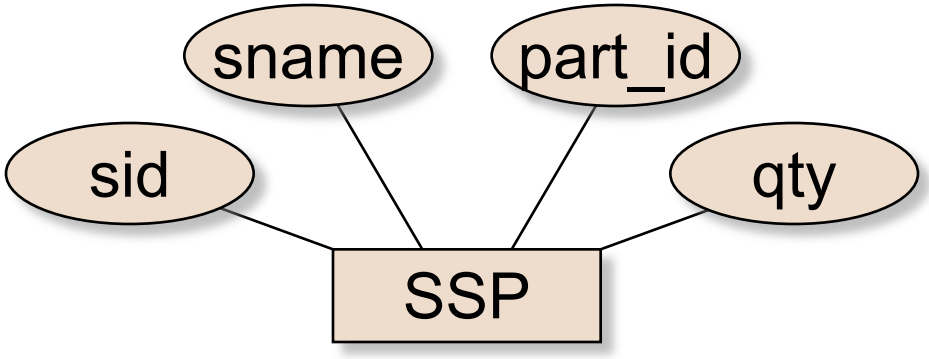
Candidate keys are (sid, part_id)
and (sname, part_id).

$\{ sid, part\_id \} \rightarrow qty$

$\{ sname, part\_id \} \rightarrow qty$

$sid \rightarrow sname$

$sname \rightarrow sid$

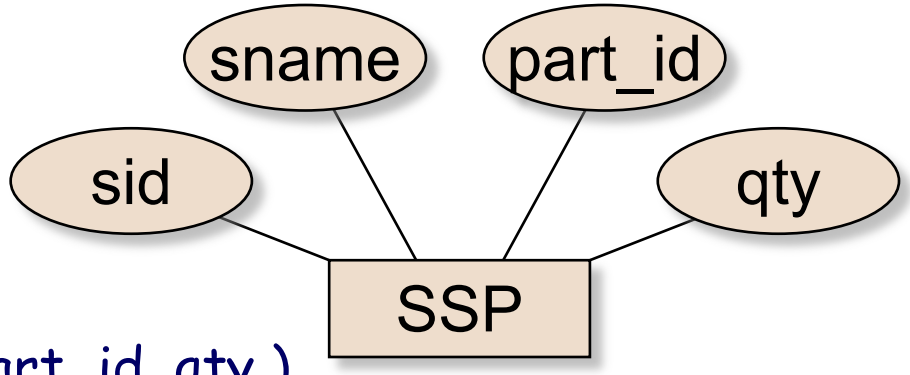


The relation is in 3NF:

For $sid \rightarrow sname$, … $sname$ is in a candidate key.

For $sname \rightarrow sid$, … sid is in a candidate key.

However, this leads to redundancy.



If we decompose the schema into

R_1 = (sid, sname), R_2 = (sid, part_id, qty)

These are in BCNF.

The decomposition is dependency preserving.

{ *sname*, *part_id* } → *qty* can be deduced from

(1) sname → sid	(given)
(2) { sname, part_id } → { sid, part_id }	(augmentation on (1))
(3) { sid, part_id } → qty	(given)

and finally transitivity on (2) and (3).

Design Goals

- Goal for a relational database design is:
  - BCNF
  - lossless join
  - Dependency preservation

- If we cannot achieve this, we accept:
  - 3NF
  - lossless join
  - Dependency preservation