4. Relational Database Modelling and Functional Dependency

Today's Lecture

1. Problems in Relational Models
2. Duplicate vs Redundant Data
3. Normal Form Databases
4. Determinants and Identifiers
5. Determinants and Redundancy
6. Well-normalised tables
7. Fully-normalised tables
8. Multivalued Determinacy
9. Advantages of Full normalisation

Problems in Relational Modelling

Problems in direct Relational Modelling

Objectives

- Illustrate techniques to describe information in terms of table definitions and occurrences
- Guard against anomalies when we insert, delete or lose consistency of data in the tables
- How do we know our tables are correct ?

Key Points to remember about Relational Models

1. Ordering of rows is not significant
2. Ordering of columns is not significant
3. Each row/column intersection contains a single attribute value. Multiple values are not allowed
4. Each row in a table must be distinct.
   => row can always be uniquely identified by quoting an appropriate combination of attribute values

A table conforming to these restrictions is called a normalised table

Example Simple Problem

Suppose a company called TechScience, wishes to create a database for managing its engineers and the engineering projects on which they work.

Suggest a relational structure (i.e. table(s)) for storing Engineer employee ids, Engineer Employee Names, Project Ids and their Project Names.

Suppose TechEngineering has a policy that NO Engineer can work on more than one project at a time. But a Project can have multiple Engineers working on that project.

Which is the correct table structure?

Solution A:
AllInformation (EngineerId, EngineerName, Project ID, ProjectName)

Solution B:
Engineer (EngineerId,EngineerName)
Project (Project ID, ProjectName, EngineerId)

Solution C:
Engineer (EngineerId,EngineerName, Project ID)
Project (Project ID, ProjectName)

Solution D:
Engineer (EngineerId,EngineerName)
Project (Project ID, ProjectName)
EngineerAllocationsToProjects (EngineerId, Project ID)

Which is the correct table structure?

Suppose a company called TechEngineering, wishes to create a database for managing its engineers and the engineering projects on which they work.

Suggest a relational structure (i.e. table(s)) for storing Engineer employee ids, Engineer Employee Names, Project Ids and their Project Names.

Suppose TechEngineering has a policy that NO Engineer can work on more than one project at a time. But a Project can have multiple Engineers working on that project.

Solution A:

AllInformation (EngineerId, EngineerName, Project ID, ProjectName)

Solution B:

Engineer (EngineerId, EngineerName)

Project (Project ID, ProjectName, EngineerId)

Solution C:

Engineer (EngineerId, EngineerName, Project ID)

Project (Project ID, ProjectName)

Solution D:

Engineer (EngineerId, EngineerName)

Project (Project ID, ProjectName)

EngineerAllocationsToProjects (EngineerId, Project ID)

Solution A: (as a single table)

AllInformation (EngineerId, EngineerName, Project ID, ProjectName)

EngineerId	EngineerName	Project ID	ProjectName
007	James Bond	888	Ventilator
008	Peter Parker	888	Ventilator
009	Bruce Wayne	999	Heart Monitor

Is there any **Unnecessary Duplication**??

Unnecessary Duplication: If we can delete a value in a cell, and still be able to work out What they value was, then that value was 'unnecessarily duplicated'.
Unnecessary duplication creates problems when updating data as you need to ensure that all copies of such information are updated when their value(s) are changed!

Solution A (as a single table)

Solution A:

AllInformation (EngineerId, EngineerName, Project ID, ProjectName)

EngineerId	EngineerName	Project ID	ProjectName
007	James Bond	888	Ventilator
008	Sarah-Jane Parker	888	~~Ventilator~~
009	Bruce Wayne	999	Heart Monitor

Is there any **Unnecessary Duplication**?? **YES**

As Solution B (as two tables) tables

Solution B:

EngineerId	EngineerName
007	James Bond
008	Sarah-Jane Parker
009	Bruce Wayne

ProjectId	Project Name	EngineerId
888	Ventilator	007
888	Ventilator	008
999	Heart Monitor	009

Unnecessary Duplication??

Solution B (as two tables)

Solution B:

EngineerId	EngineerName
007	James Bond
008	Sarah-Jane Parker
009	Bruce Wayne

ProjectId	Project Name	EngineerId
888	Ventilator	007
888	~~Ventilator~~	008
999	Heart Monitor	009

Is there any **Unnecessary Duplication**?? **YES**

Solution C (as two tables)

Solution C:

EngineerId	EngineerName	ProjectId
007	James Bond	888
008	Sarah-Jane Parker	888
009	Bruce Wayne	999

ProjectId	ProjectName
888	Ventilator
999	Peter Parker

Unnecessary Duplication??

Solution C (as two tables)

Solution C:

EngineerId	EngineerName	ProjectId
007	James Bond	888
008	Sarah-Jane Parker	888
009	Bruce Wayne	999

ProjectId	ProjectName
888	Ventilator
999	Heart Monitor

Unnecessary Duplication?? NO

Solution D (as three tables)

Solution C:

EngineerId	EngineerName
007	James Bond
008	Sarah-Jane Parker
009	Bruce Wayne

ProjectId	ProjectName
888	Ventilator
999	Heart Monitor

EngineerId	ProjectId
007	888
008	888
009	999

Unnecessary Duplication??

Solution D (as three tables)

Solution C:

EngineerId	EngineerName
007	James Bond
008	Sarah-Jane Parker
009	Bruce Wayne

ProjectId	ProjectName
888	Ventilator
999	Heart Monitor

EngineerId	ProjectId
007	888
008	888
009	999

Unnecessary Duplication?? NO

You can't delete an attribute value in any of the tables without loosing some information from the database. i.e. you can't delete a value and then work out what that value was.

Duplicate vs Redundant Data

Duplicated vs Redundant Data

- Must be careful to distinguish between redundant and duplicated data

- **Duplicated data**:- occurs where an attribute (column) has two or more identical values

- **Redundant data**:- occurs if you can delete a value without information being lost

  => redundancy is unnecessary duplication

Duplication vs. Redundancy

Example

Part#	Part Desc
P2	Nut
P1	Bolt
P3	Washer
P4	Nut

- Parts and their textual description are contained in the table
- We can see some information is duplicated (nut)
- Is there any *redundant* information here?

Duplication vs. Redundancy

Example

Part#	Part Desc
P2	Nut
P1	Bolt
P3	Washer
P4	Nut

delete nut

loss of information

Part#	Part Desc
P2	…
P1	Bolt
P3	Washer
P4	Nut

=> *nut* was duplicate but NOT redundant!

Duplicated vs. Redundant

S#	Part#	Part Desc
S2	P1	Bolt
S7	P6	Bolt
S2	P4	Nut
S5	P1	Bolt

- Now indroduce S# supplier ID
- Table represents the following information
  - Part ID and its description

    AND
  - Who supplies each part

Note: two suppliers can supply the same part

We can see duplicate information (bolt x 3)

Is any of the information here redundant?

Duplicated vs. Redundant (3)

- Now introduce S# supplier ID

S#	Part#	Part Desc
S2	P1	Bolt
S7	P6	Bolt
S2	P4	Nut
S5	P1	Bolt

Delete bolt

no loss of information

S#	Part#	Part Desc
S2	P1	Bolt
S7	P6	Bolt
S2	P4	Nut
S5	P1	…

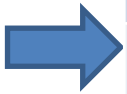
=> Some duplicated data was redundant

Eliminating Redundancy

- We cannot just delete values from the table in the previous example!

- Preferable to split table into 2 tables

S#	Part#	Part Desc
S2	P1	Bolt
S7	P6	Bolt
S2	P4	Nut
S5	P1	…



Part#	Part Desc
P1	Bolt
P6	Bolt
P4	Nut
P1	…

S#	Part#
S2	P1
S7	P6
S2	P4
S5	P1

Eliminating Redundancy

- Eliminated redundancy by table splitting
  - P1 description only appears once
  - relationship is made by including part# in two tables
- So far we've assumed that table structures which permit redundancy can be recognised by inspection of table occurrence
- This is not entirely accurate since attribute values are subject to insertion / change / deletion

Eliminating Redundancy

SP

S#	Part#	Part Desc
S2	P1	Bolt
S7	P6	Bolt
S2	P4	Nut

Inspection of table SP does not reveal any redundancy

Could even suggest that
"no two suppliers may supply same part#"

Repeating Groups

- We stated earlier that:

  " Each attribute must have at most one value in each row"

S#	Sname	P#
S5	Wells	P1
S2	Heath	P1, P4
S7	Barron	P6
S9	Edwards	P8, P2, P6

Repeating Groups

- We stated earlier that:

  " Each attribute must have at most one value in each row"

S#	Sname	P#
S5	Wells	P1
S2	Heath	P1, P4
S7	Barron	P6
S9	Edwards	P8, P2, P6

Not possible

Repeating Groups (2)

Problems:

1. Table is asymmetric representation of symmetrical data

2. Rows can be sorted into s# but not into p#

3. Rows are different length due to variation in number of p#'s

4. If rows were fixed length, they would need to be padded with null values

Elimination of Repeating Groups

- Easiest way to eliminate repeating groups is to write out the table occurrence using a vertical layout and fill in the blanks by duplicating the non- repeating data necessary

S#	SName	P#
S5	Wells	P1
S2	Heath	P1
		P4
S9	Edwards	P8
		P2
		P6

S#	SName	P#
S5	Wells	P1
S2	Heath	P1
S2	Heath	P4
S9	Edwards	P8
S9	Edwards	P2
S9	Edwards	P6

But this can lead to ‘redundancy’ of information!
(Does the right hand table contain redundant information?)

Elimination of Repeating Groups(2)

Alternate method:

- Split table into two tables so that repeating group appears in one table and rest of attributes in another

- Need to provide correspondence between tables by including a key attribute with the repeating group table

S#	SName
S5	Wells
S2	Heath
S7	Barron
S9	Edwards

S#	P#
S5	P1
S2	P1
S2	P4
S7	P6
S9	P8
S9	P2
S9	P6

Eliminating Repeating Groups & Redundancy

- Snapshot of table is inadequate guide to presence / absence of redundant data
- Need to know underlying rules
- DBA must discover rules which apply to conceptual model

Unfortunate Conclusion

- It is not possible to tell by looking at the relational tables in a DB to determine if
  - There is the potential for redundancy

- But what would be a ‘correctly formed’ table?

Normal Form Databases

Codd's Normal Forms

- Codd identified some rules which govern the way we create tables to avoid anomalies when inserting or deleting values in these tables.
- These rules are called NORMAL forms. There are three and a half important levels (and two further levels which are occasionally used)

- 1st Normal Form:
  - A relation is in first normal form if the domain of each attribute contains only atomic values and the value of each attribute contains only a single value from that domain
- 2nd Normal Form
  - A relation is in 2nd normal form if, in addition to satisfying the criteria for 1st normal form, every non-key column is *fully functionally dependent* on the entire primary key.
- 3rd Normal Form
  - A relation is in 3rd Normal Form if, in addition to satisfying the criteria for 2nd Normal Form, and no non-key attributes are *transitively dependent* upon the primary key
- Boyce Codd Normal Form (also called 3 ½ Normal Form)
  - "all attributes in a relation should be dependent on the key, the whole key and nothing but the key

Summary 1NF – 3NF

Normal Form	Test	Remedy (Normalization)
First (1NF)	Relation should have no multivalued attributes or nested relations.	
Second (2NF)		
Third (3NF)		

Summary 1NF – 3NF

Normal Form	Test	Remedy (Normalization)
First (1NF)	Relation should have no multivalued attributes or nested relations.	Form new relations for each multivalued attribute or nested relation.
Second (2NF)		
Third (3NF)		

Summary 1NF – 3NF

Normal Form	Test	Remedy (Normalization)
First (1NF)	Relation should have no multivalued attributes or nested relations.	Form new relations for each multivalued attribute or nested relation.
Second (2NF)	For relations where primary key contains multiple attributes, no nonkey attribute should be functionally dependent on a part of the primary key.	
Third (3NF)		

Summary 1NF – 3NF

Normal Form	Test	Remedy (Normalization)
First (1NF)	Relation should have no multivalued attributes or nested relations.	Form new relations for each multivalued attribute or nested relation.
Second (2NF)	For relations where primary key contains multiple attributes, no nonkey attribute should be functionally dependent on a part of the primary key.	Decompose and set up a new relation for each partial key with its dependent attribute(s). Make sure to keep a relation with the original primary key and any attributes that are fully functionally dependent on it.
Third (3NF)		

Summary 1NF – 3NF

Normal Form	Test	Remedy (Normalization)
First (1NF)	Relation should have no multivalued attributes or nested relations.	Form new relations for each multivalued attribute or nested relation.
Second (2NF)	For relations where primary key contains multiple attributes, no nonkey attribute should be functionally dependent on a part of the primary key.	Decompose and set up a new relation for each partial key with its dependent attribute(s). Make sure to keep a relation with the original primary key and any attributes that are fully functionally dependent on it.
Third (3NF)	Relation should not have a nonkey attribute functionally determined by another nonkey attribute (or by a set of nonkey attributes). That is, there should be no transitive dependency of a nonkey attribute on the primary key.	

Summary 1NF – 3NF

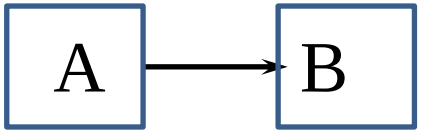
Normal Form	Test	Remedy (Normalization)
First (1NF)	Relation should have no multivalued attributes or nested relations.	Form new relations for each multivalued attribute or nested relation.
Second (2NF)	For relations where primary key contains multiple attributes, no nonkey attribute should be functionally dependent on a part of the primary key.	Decompose and set up a new relation for each partial key with its dependent attribute(s). Make sure to keep a relation with the original primary key and any attributes that are fully functionally dependent on it.
Third (3NF)	Relation should not have a nonkey attribute functionally determined by another nonkey attribute (or by a set of nonkey attributes). That is, there should be no transitive dependency of a nonkey attribute on the primary key.	Decompose and set up a relation that includes the nonkey attribute(s) that functionally determine(s) other nonkey attribute(s).

Determinants and Identifiers

Determinants

- If there are rules such that duplicate values of attribute A are always associated with the *same* value of attribute B (within any given occurrence of the table) then attribute A is a determinant of attribute B

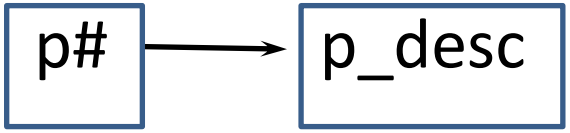
Note: A is determinant of B can be written as



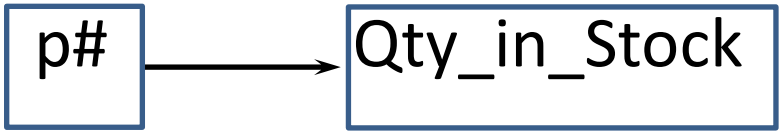
Determinants (2)

Example:

- If each possible p# value has precisely one associated part description value (i.e. P4 has just one description nut) then we can say that p# is a determinant of part description



- Similarly, if each possible p# value has precisely one quantity in stock then we can say p# is a determinant of quantity in stock



Determinants

Stock

P#	P_desc	Qty
P2	Nut	5000
P1	Bolt	8300
P3	Washer	9750
P4	Nut	2326

Question:

is P_Desc a determinant of P# ?

is P_Desc a determinant of Qty ?

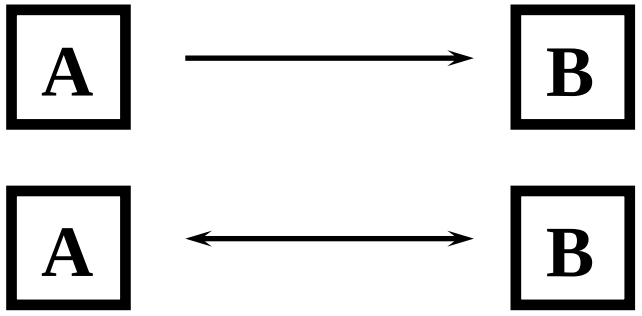
Superfluous Attribute

- If P# determines Qty then composite attribute {P#, P_Desc} also determines Qty, but P_Desc is superfluous

- We assume determinants do not contain any superfluous attributes

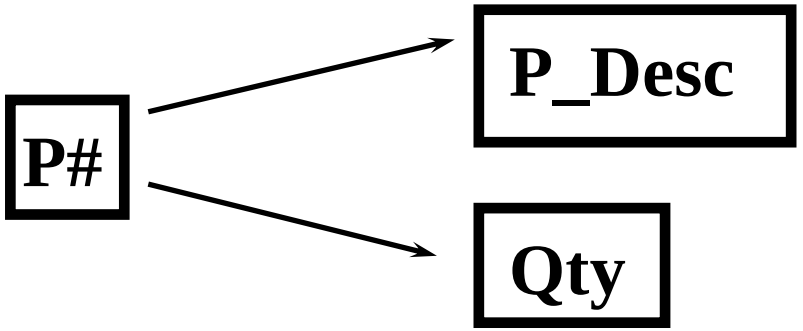
Determinacy Diagrams

- We need a notation to express where one attribute determines another => we use determinacy diagrams



A is determinant of B

A is determinant of B and vice versa



Determinacy diagram of stock table

Determinacy Diagrams (2)

EXAMPLE – Enterprise Rules as follows

- Supplier identified by single S# & a part is identified by single P#
- Each supplier has only one SName but different suppliers may have same names
- A given supplier supplies a given part in just one pack size
- A supplier may supply many different parts in many different pack sizes

Determinacy Diagrams (2)

EXAMPLE

- Supplier identified by single S# & a part is identified by single P#



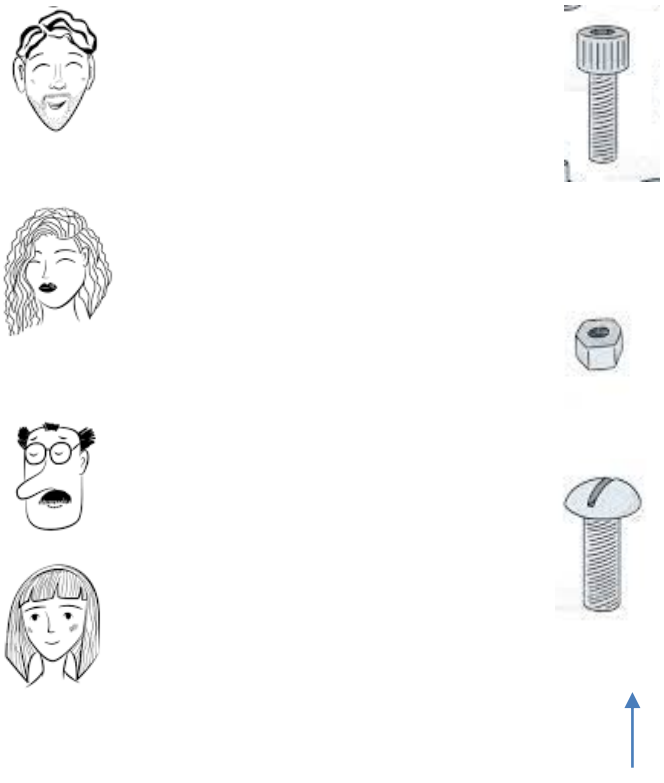




Determinacy Diagrams (2)

EXAMPLE

- Supplier identified by single S# & a part is identified by single P#



Determinacy Diagrams (2)

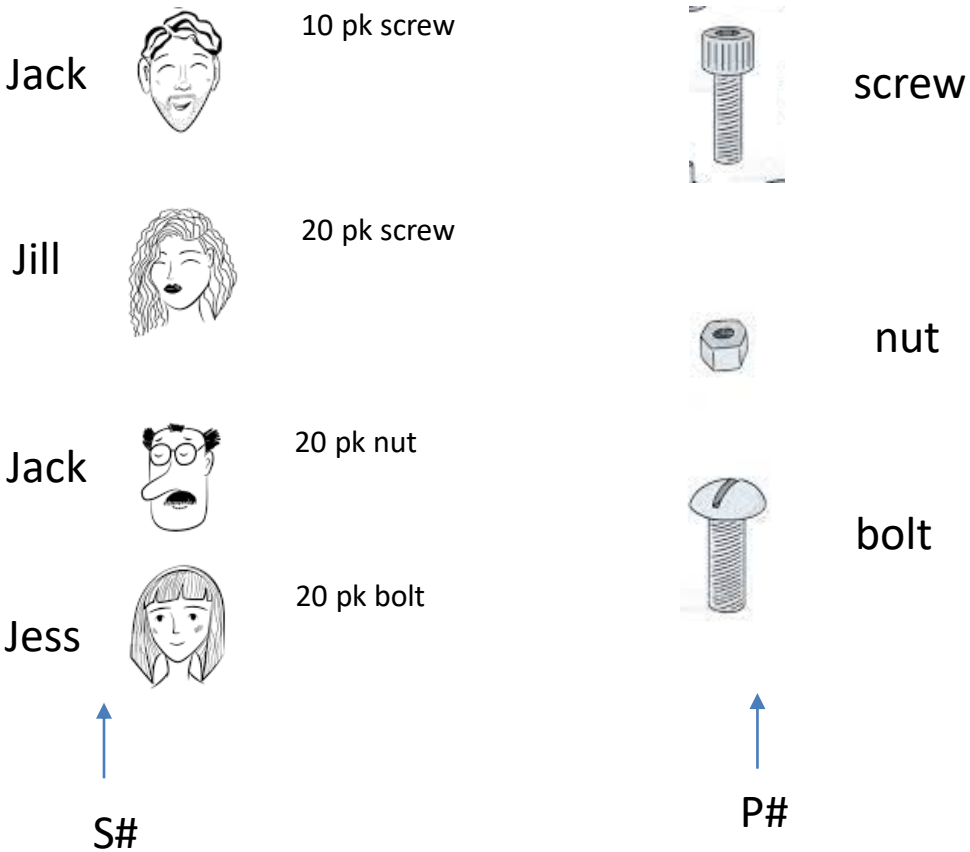
EXAMPLE

- Each supplier has only one SName but different suppliers may have same names

Determinacy Diagrams (2)

EXAMPLE

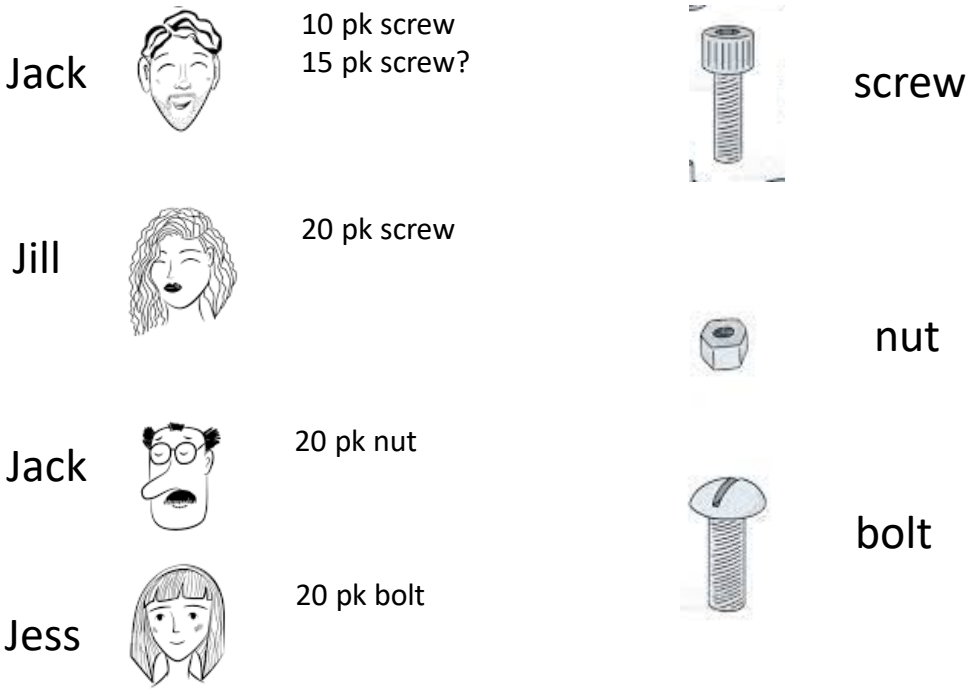
- A given supplier supplies a given part in just one pack size



Determinacy Diagrams (2)

EXAMPLE

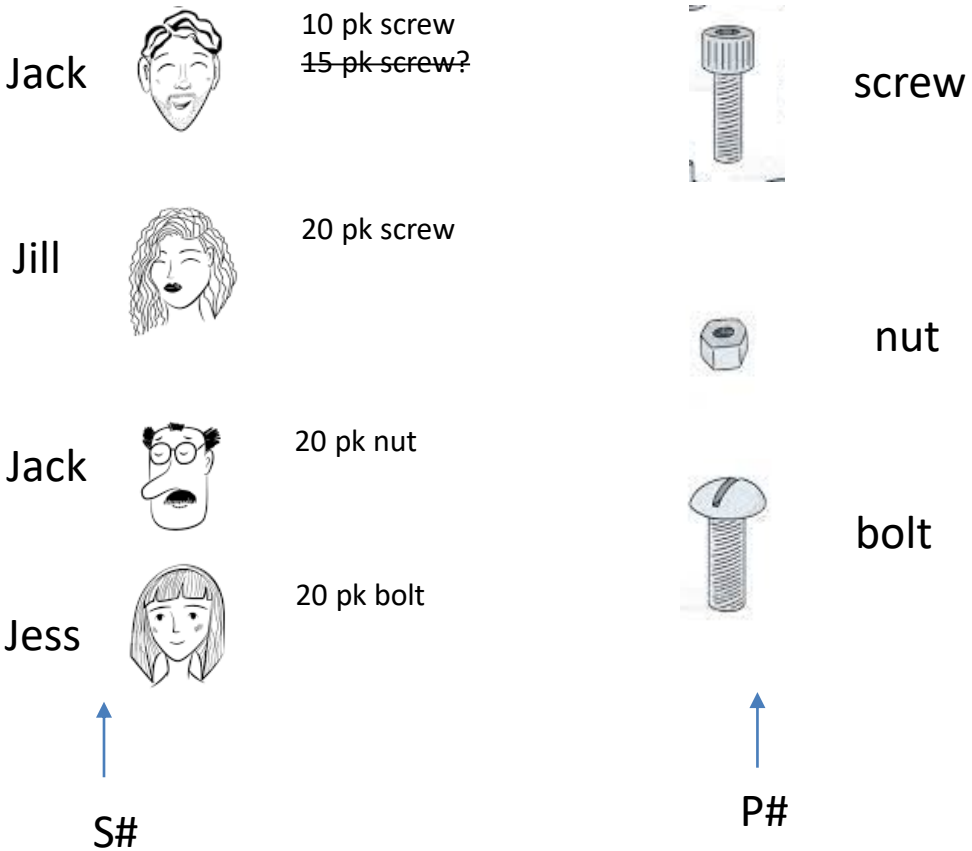
- A given supplier supplies a given part in just one pack size



Determinacy Diagrams (2)

EXAMPLE

- A given supplier supplies a given part in just one pack size



Determinacy Diagrams (2)

EXAMPLE

- A supplier may supply many different parts

Jack 
10 pk screw
20 pk nut

Jill 
20 pk screw

Jack 
20 pk nut
15 pk bolt

Jess 
15 pk screw
10 pk nut
20 pk bolt


screw


nut


bolt

S#

P#

Determinacy Diagrams (2)

EXAMPLE

- A supplier may supply many different parts

Jack 
10 pk screw
20 pk nut

Jill 
20 pk screw

Jack 
20 pk nut
15 pk bolt

Jess 
15 pk screw
10 pk nut
20 pk bolt
15 pk bolt?

S#


screw


nut


bolt

P#

Determinacy Diagrams (2)

EXAMPLE

- A supplier may supply many different parts

Jack 
10 pk screw
20 pk nut

Jill 
20 pk screw

Jack 
20 pk nut
15 pk bolt

Jess 
15 pk screw
10 pk nut
20 pk bolt
~~15 pk bolt?~~

S#


screw


nut


bolt

P#

Determinacy Diagrams (2)

EXAMPLE

- Supplier identified by single S# & a part is identified by single P#
- Each supplier has only one SName but different suppliers may have same names
- A given supplier supplies a given part in just one pack size
- A supplier may supply many different parts in many different pack sizes

Jack 

10 pk screw
20 pk nut

Jill 

20 pk screw

Jack 

20 pk nut
15 pk bolt

Jess 

15 pk screw
10 pk nut
20 pk bolt

S#

screw

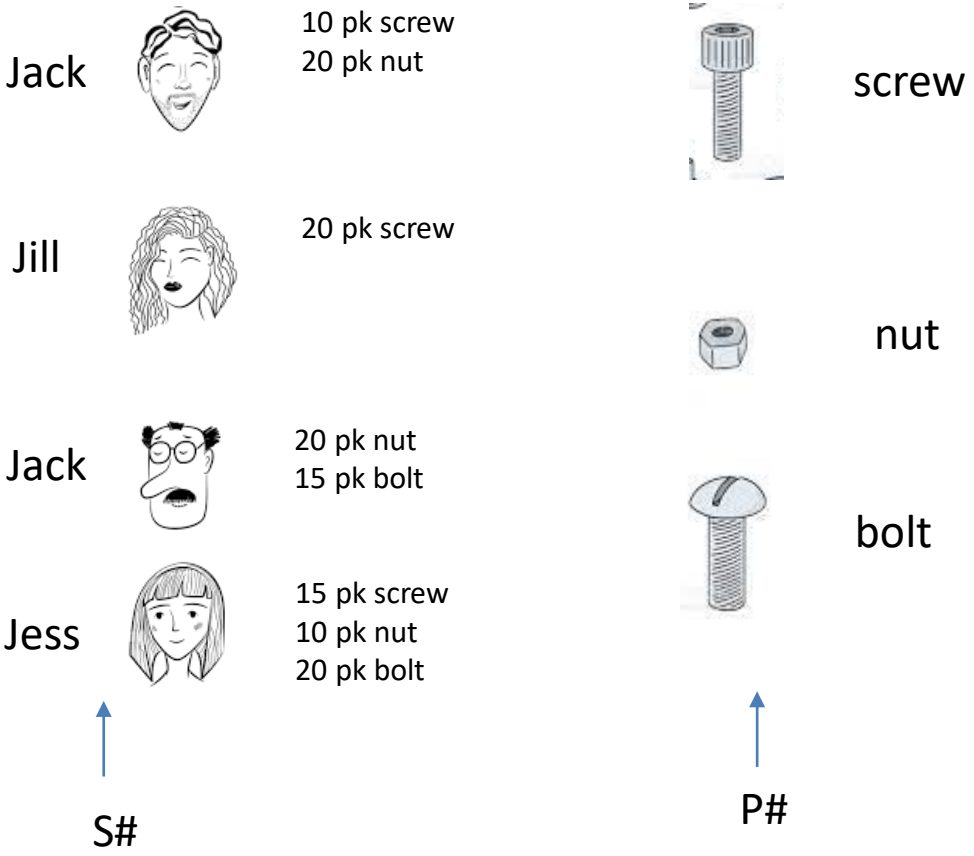
nut

bolt

P#

Determinacy Diagrams (3)

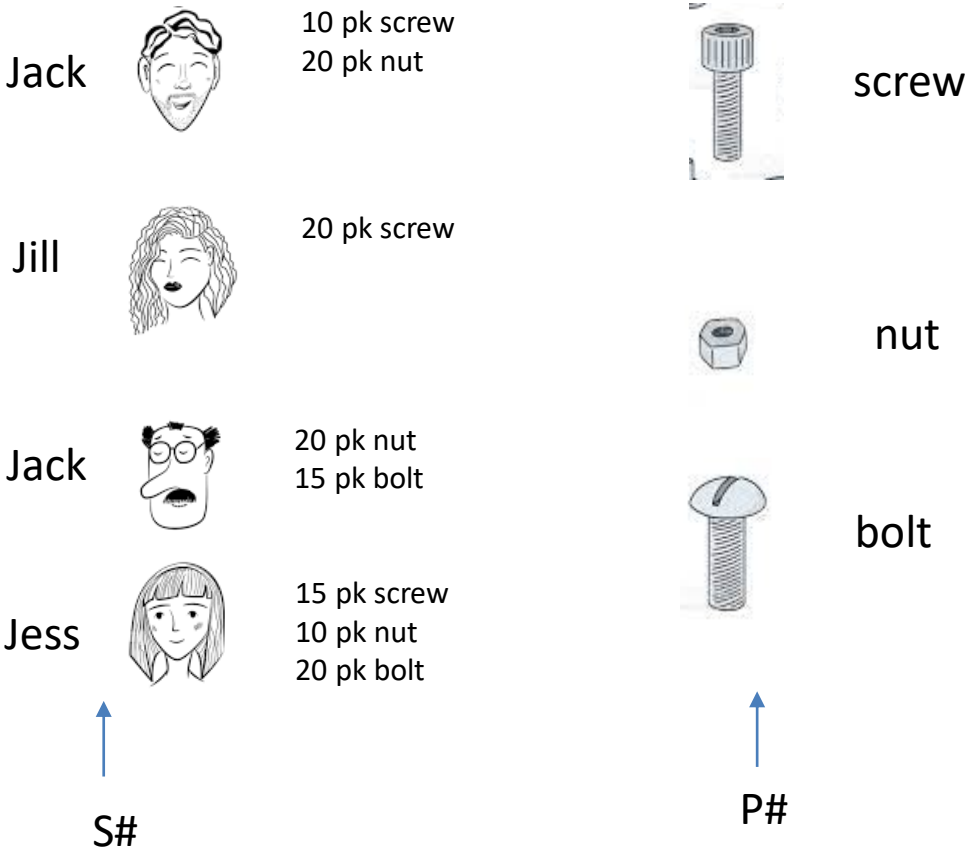
- Can recognise determinants by drawing determinacy diagrams
- What determiners **Sname**?



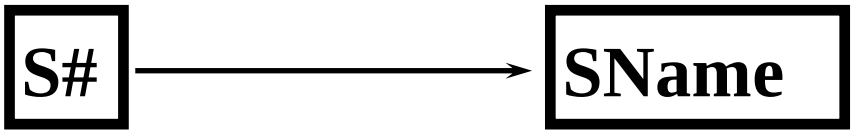
? → **SName**

Determinacy Diagrams (3)

- Can recognise determinants by drawing determinacy diagrams
- What determiners **Sname**?



S# is a determinant for SName

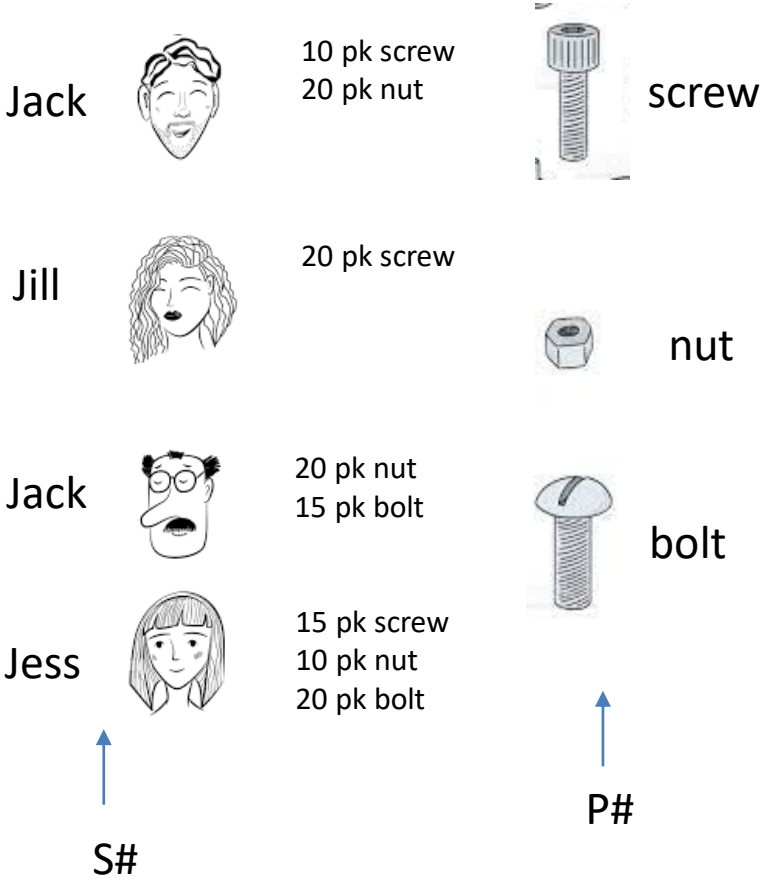


Determinacy Diagrams (3)

- Can recognise determinants by drawing determinacy diagrams
- What determiners **Packsize**?

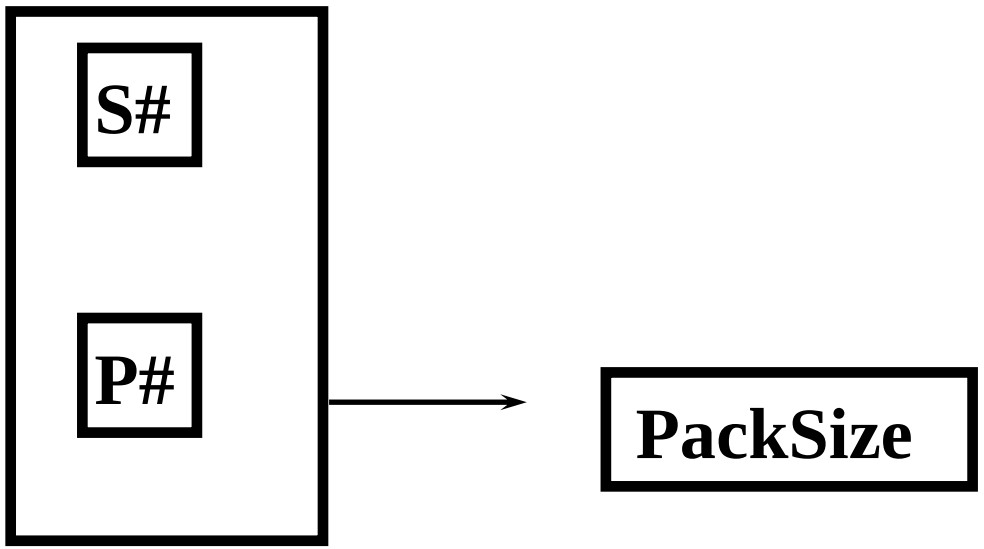
?

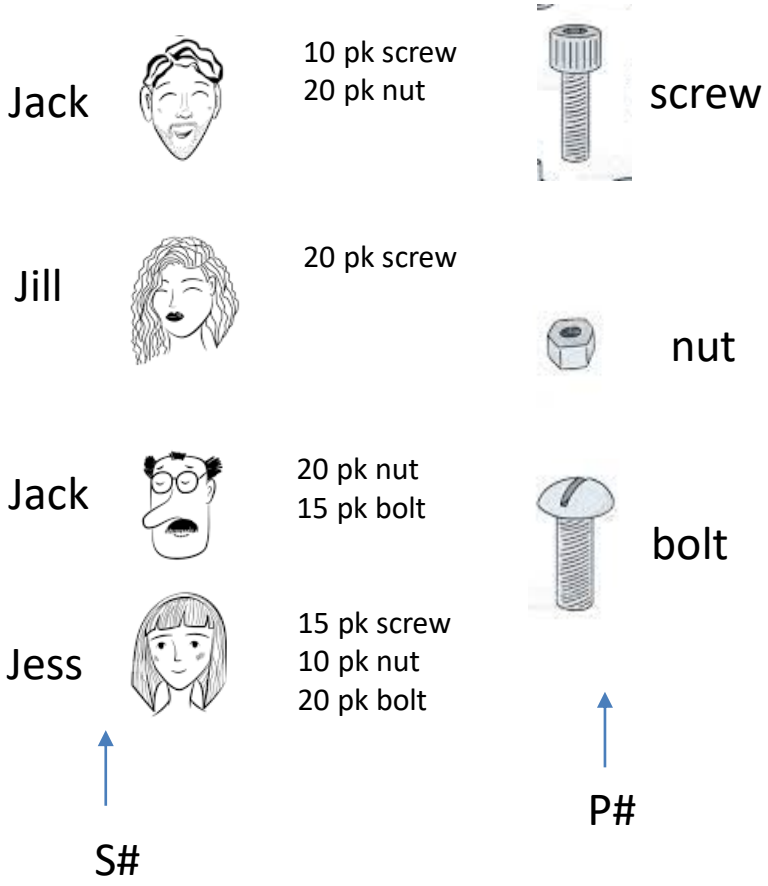
PackSize



Determinacy Diagrams (3)

- Can recognise determinants by drawing determinacy diagrams
- What determiners **Packsize**?

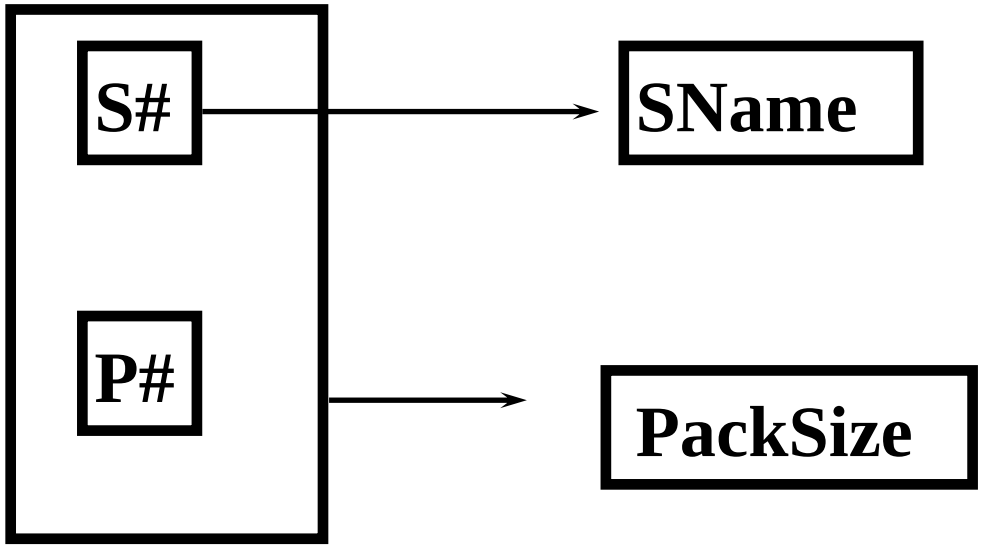


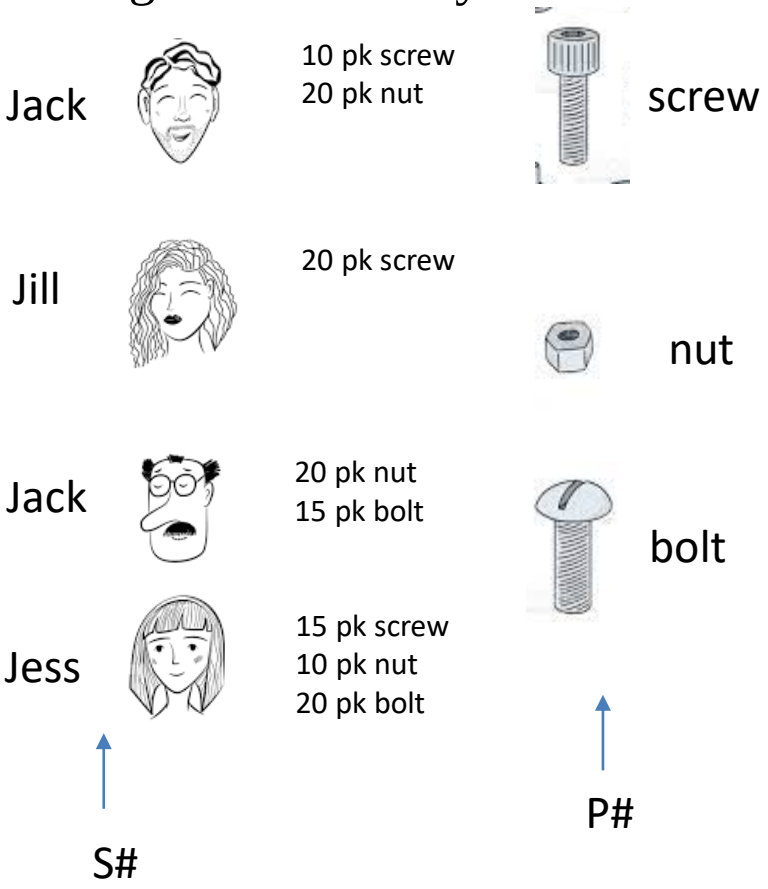


{ S#, P# } is determinant for packsize

Determinacy Diagrams (3)

- Can recognise determinants by drawing determinacy diagrams
- Combine determinants for SName and Packsize in single Determinacy Diagram



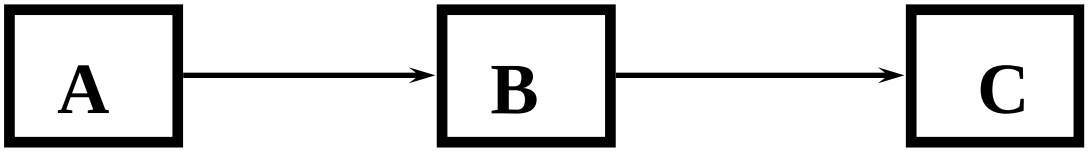


S# is determinant for SName
{ S#, P# } is determinant for packsize

Transitive Determinants

- If A is determinant of B & B is determinant of C

  Then A is determinant of C



Identifiers

- Because of the rule that 'no two rows in a table can have identical values throughout'

<u>therefore</u>

individual row can always be identified by quoting the values of all its attributes. However some values may not be needed.

Identifiers

Example:

Employee (Employee#, Employee_name, Salary)

Identifiers

Example:

Employee (Employee#, Employee_name, Salary)

Jack  30K

Jill  20 K

Jack  25K

Jess  45 K

Identifiers

Example:

Employee (Employee#, Employee_name, Salary)

Rules:
• No two rows should have the same value for Employee#

=> Employee# is a row identifier of the table

Jack  30K

Jill  20 K

Jack  25K

Jess  45 K

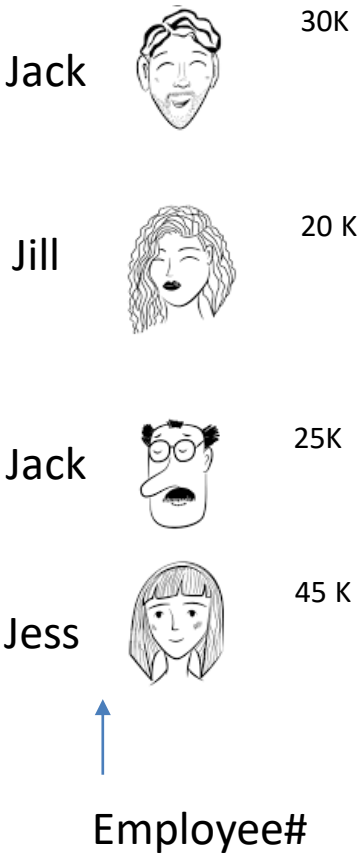
Identifiers

Example:

Employee (Employee#, Employee_name, Salary)

Rules:
• No two rows should have the same value for Employee#

=> Employee# is a row identifier of the table



Identifiers

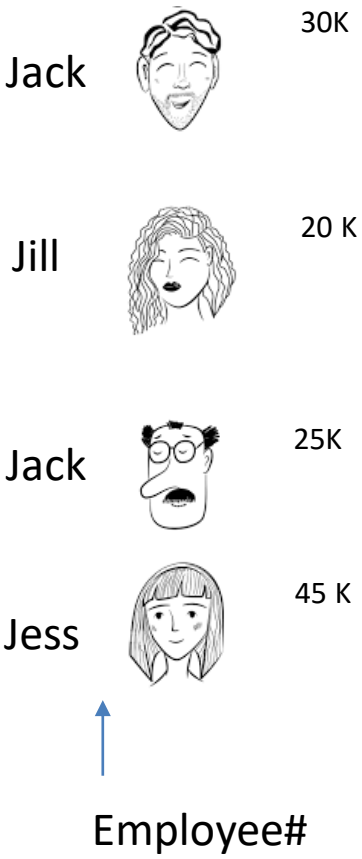
Example:

Employee (Employee#, Employee_name, Salary)

Rules:
• No two rows should have the same value for Employee#

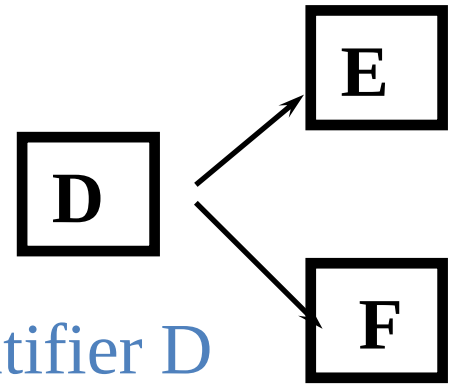
=> Employee# is a row identifier of the table

• Note: where a composed attribute forms the identifier
=> no component part (of identifier) can be null
(entity constraint)

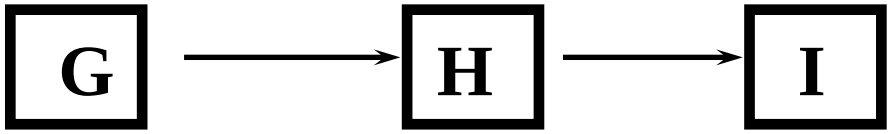


Identifiers

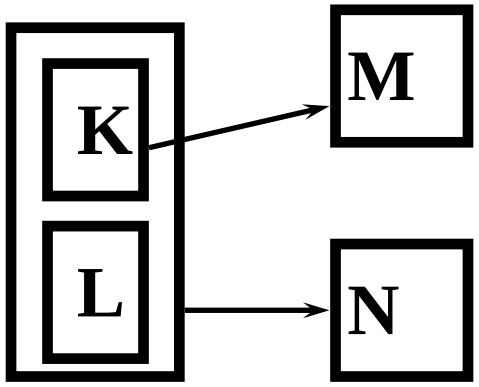
Examples:



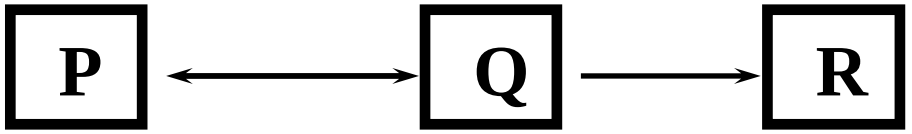
Identifier D



Identifier G



Identifier {K, L}

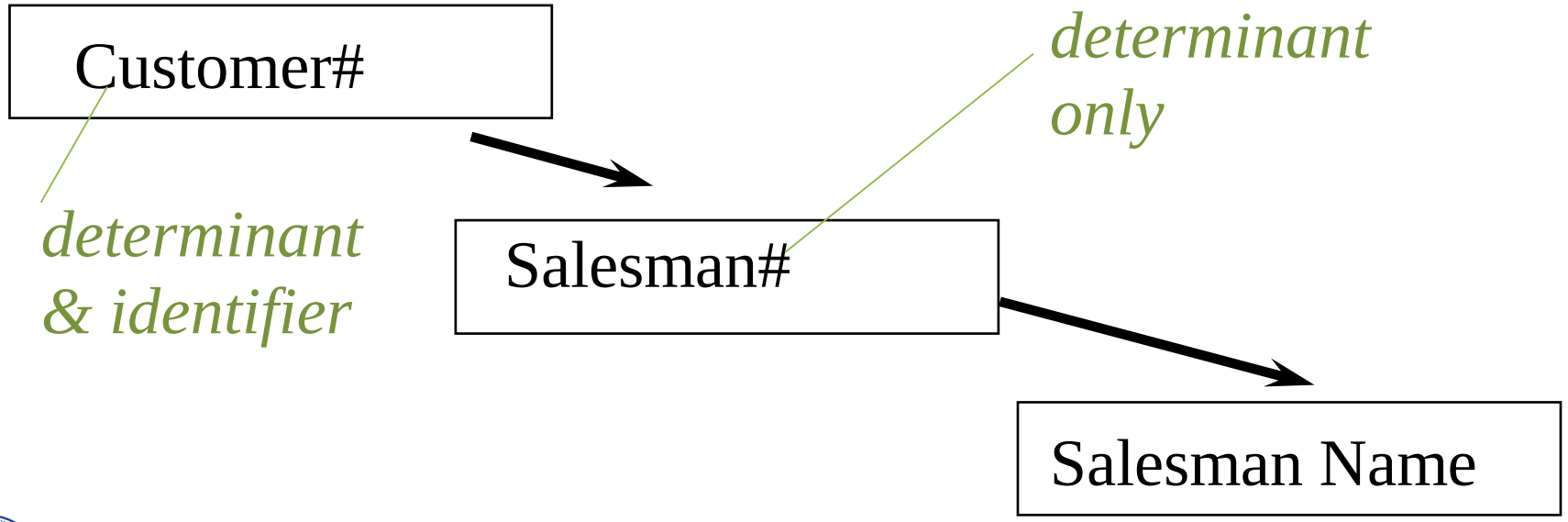


Identifier P & Q
2 candidate identifiers

Determinants and Redundancy

Determinacy & Redundancy

- Given a determinacy diagram we can detect and eliminate table structures which could contain redundant data



Determinacy & Redundancy

- Each customer# is associated with one salesman# but a salesman# may be associated with several different customer#
- Therefore salesman# could have duplicate values
- But salesman# is a determinant of salesman name
- Therefore each occurrence of a duplicate salesman# value will be associated with the same salesman name => table can contain redundant values of salesman

Customer#	Salesman#	Salesman_name
1	21	John
2	25	Jack
3	29	Jim
4	29	Jim

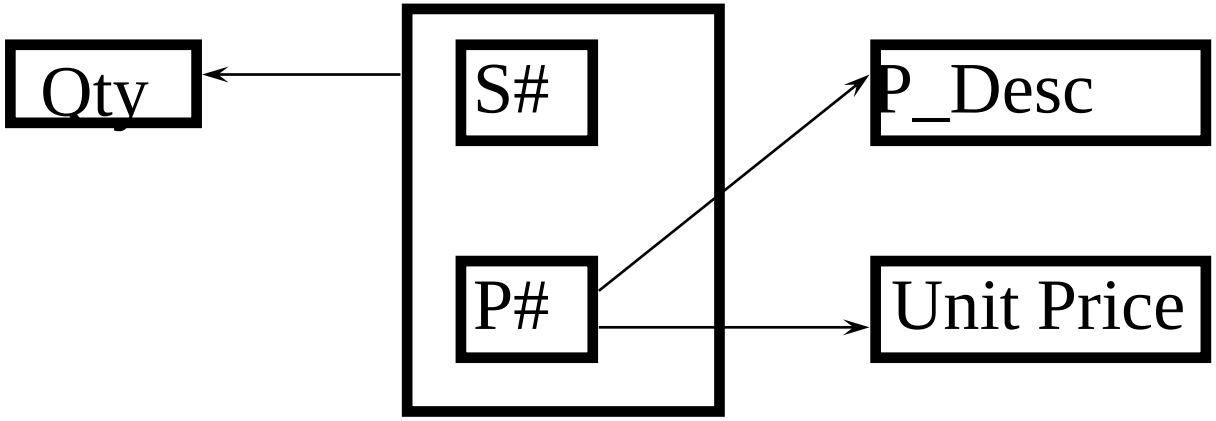
Determinacy & Redundancy

- But customer# values cannot be duplicated (because customer# is the identifier for our table) so we cannot allow redundant values of salesman#

- Potential redundancy arises because salesman# is a determinant but not a candidate identifier (salesman# determines customers but does there's a better way to identify them)

Determinacy & Redundancy

Example 2: Parts and suppliers



Potential redundancy in values of P_desc and Unit_Price
P# is a determinant but not a candidate identifier! Gives rise to Boyce-Codd Rule for detecting redundancy

Well-normalised tables

Transforming table into *well-normalised* table

- Boyce/Codd rule for determining redundancy is rule "Every determinant must be a candidate identifier"

- A table which obeys this rule is said to be in Boyce / Codd normal form (BCNF)

to put it another way:
"all attributes in a relation should be dependent on the key, the whole key and nothing but the key"

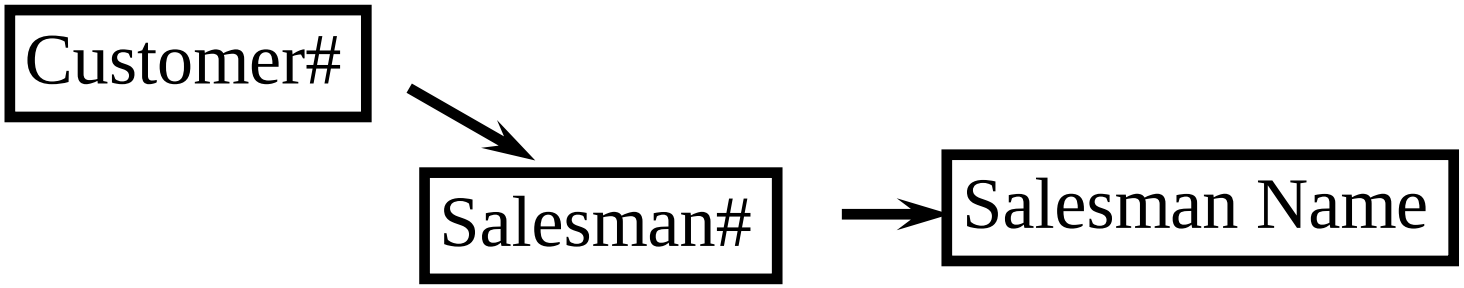
Transformation into Well N.F.

• A determinant which is *not a candidate identifier* is called a non identifying determinant

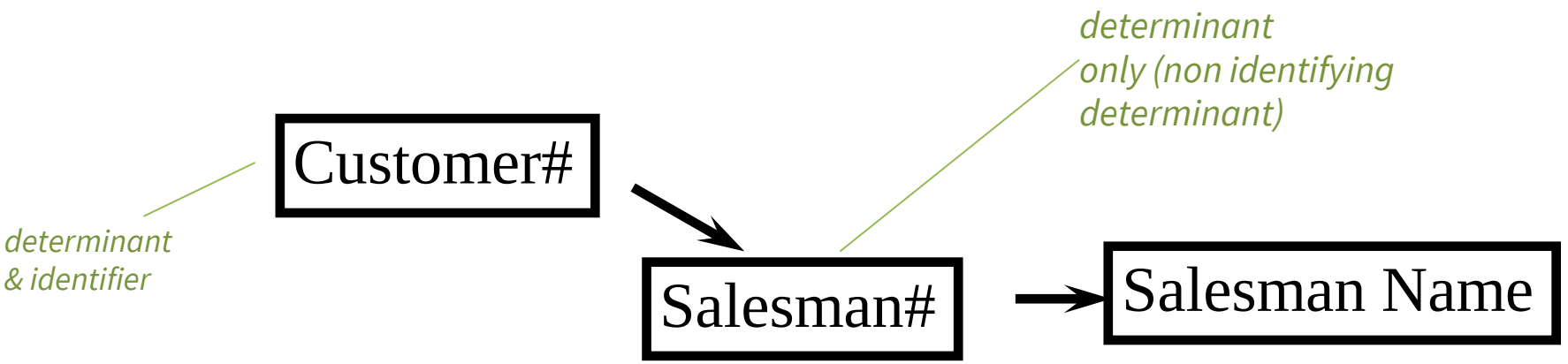
• To transform a badly normalised (non BCNF) table into well normalised tables:

Create new tables such that each non identifying determinant in the old table becomes a candidate identifier in a new table

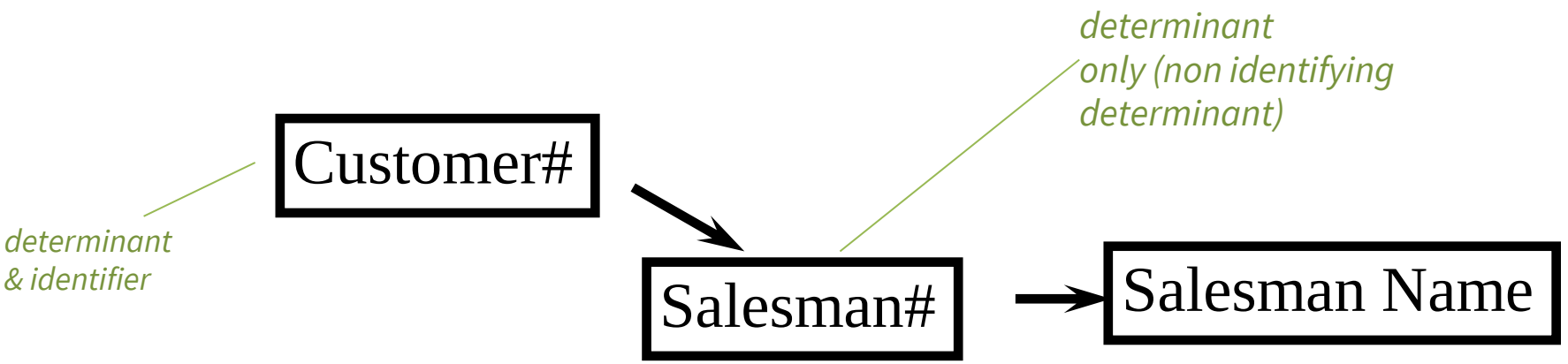
Example of BCNF Normalisation



Example of BCNF Normalisation



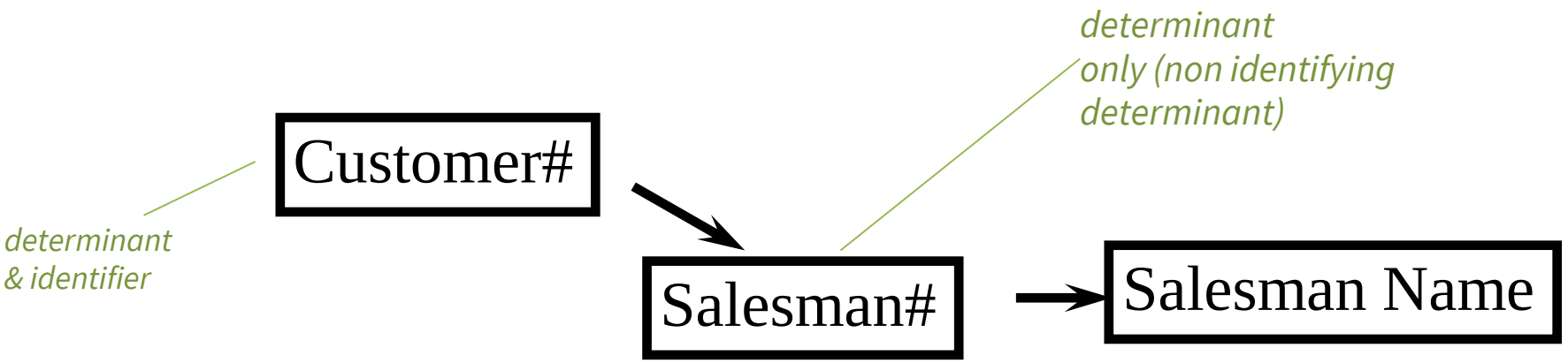
Example of BCNF Normalisation



Potential Table:

Customer#	Salesman#	Salesman_Name
3	A	John
4	A	John
5	B	Jack
6	C	Jim

Example of BCNF Normalisation

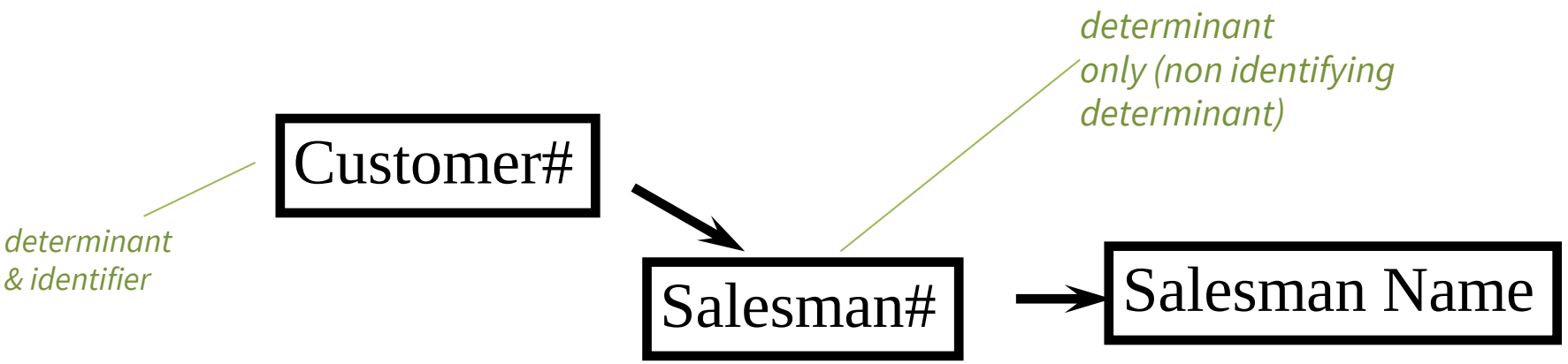


Potential Table:

Customer#	Salesman#	Salesman_Name
3	A	John
4	A	John
5	B	Jack
6	C	Jim

Potential redundancy

Example of BCNF Normalisation

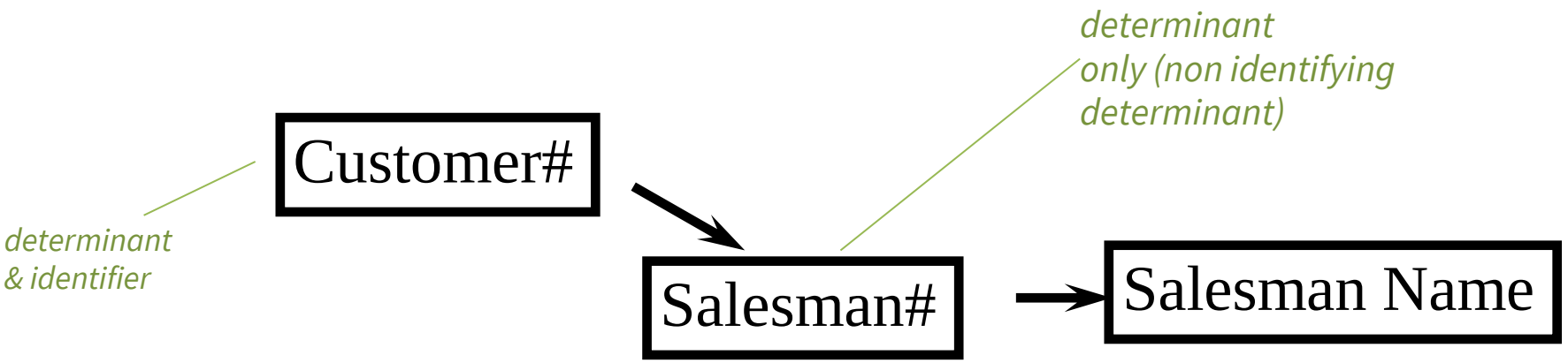


Potential Table:

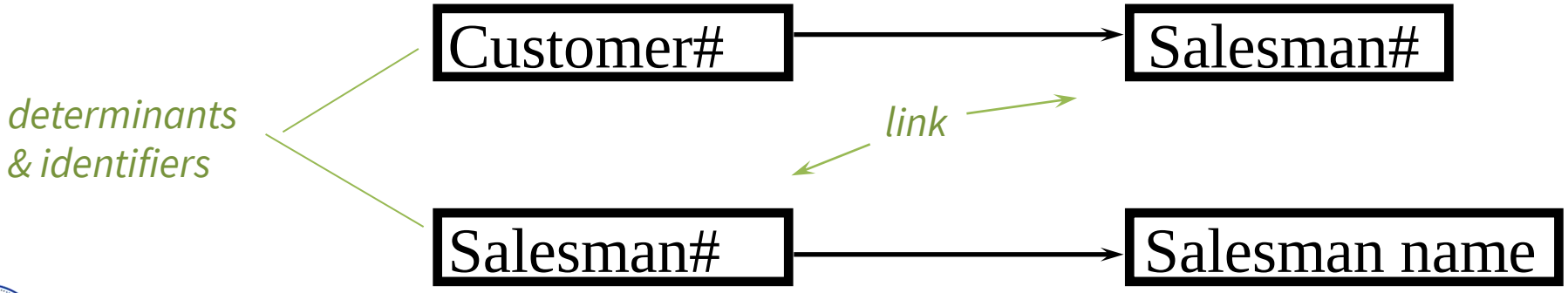
Customer#	Salesman#	Salesman_Name
3	A	John
4	A	John
5	B	Jack
6	C	Jim

Danger if we tried to update Salesman_Name

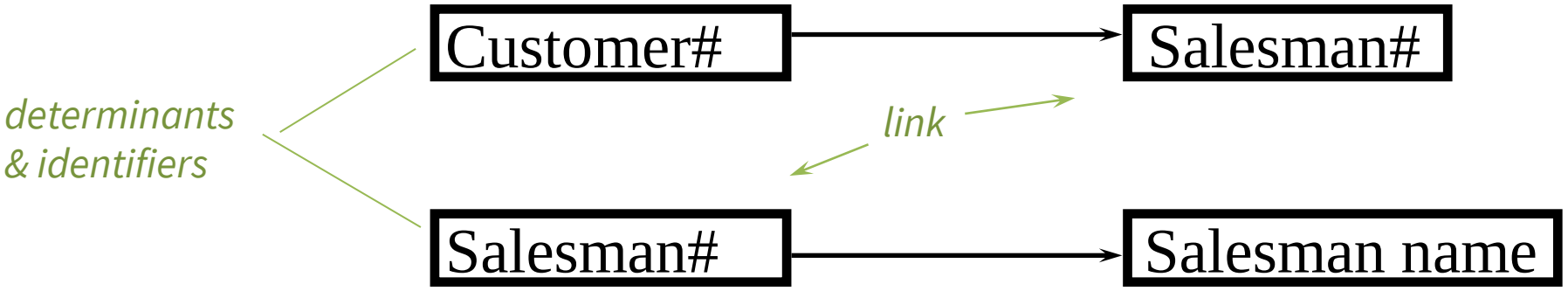
Example of BCNF Normalisation



in normalised form is:



Example of BCNF Normalisation

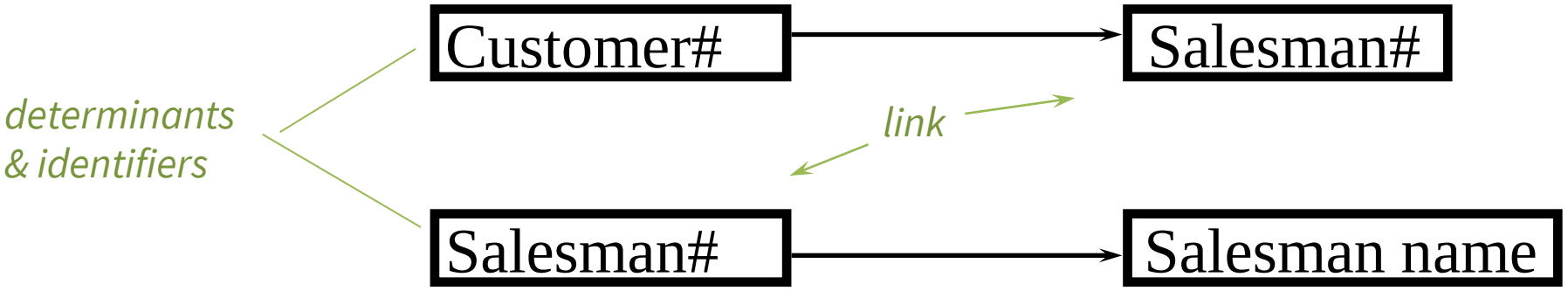


Customer#	Salesman#	Salesman_Name
3	A	John
4	A	John
5	B	Jack
6	C	Jim

Customer#	Salesman#
3	A
4	A
5	B
6	C

Salesman#	Salesman_name
A	John
B	Jack
C	Jim

Example of BCNF Normalisation



Tables transformed into well-normalised form

Customer#	Salesman#
3	A
4	A
5	B
6	C

Salesman#	Salesman_name
A	John
B	Jack
C	Jim

Fully Normalised Tables

Fully Normalised Tables

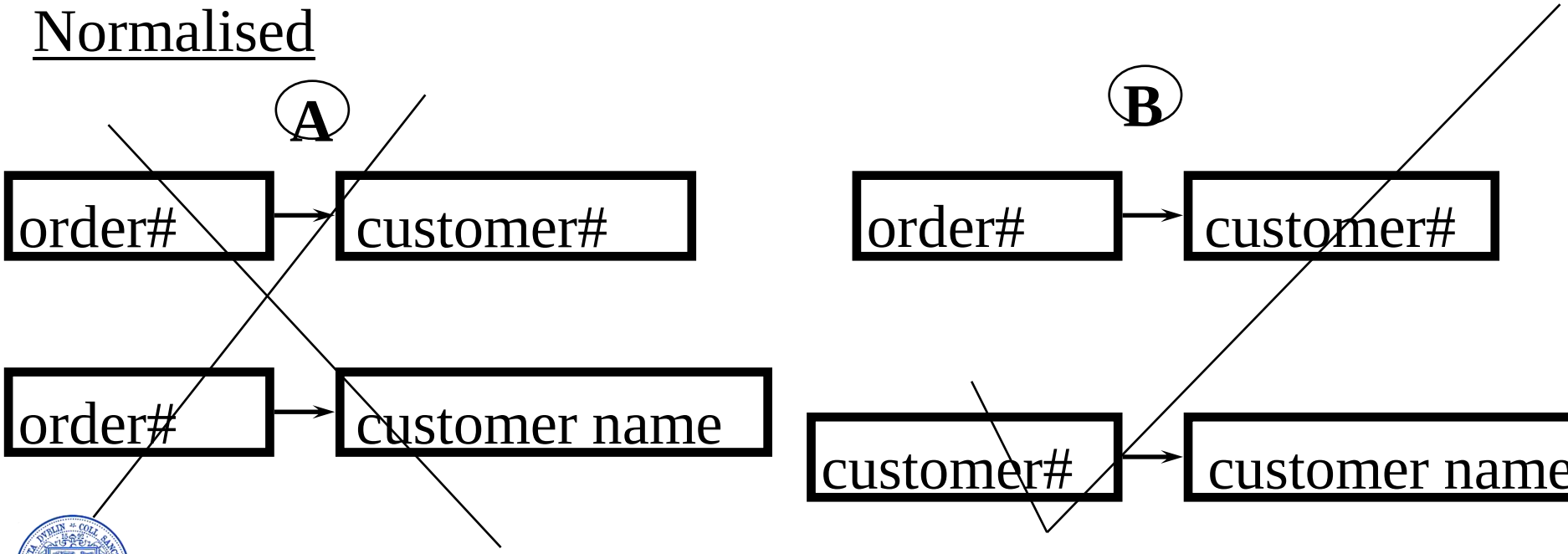
- Fully normalised tables are structured in such a way that they **cannot** contain redundant data

- Generally, a well normalised table (i.e. one in which each determinant is a candidate identifier) is also fully normalised, but not always! - so further normalisation may be desirable.

Fully Normalised Tables

Badly Normalised (hidden transitive dependency)

Normalised



Fully Normalised Tables

Ⓐ contains redundant data e.g.

Order#	Customer#
1	c1
2	c2
3	c3

Order#	Customer_name
1	Smith
2	Jones
3	Smith

- If we delete **Smith** from row 3 of **customer_name** table
  - can still use **order#**(1) to find corresponding **customer#** in **order_cust** table
  - search **order_cust** table for another **order#** placed by that customer
  - uses **order#** to search **customer_name** table for corresponding **customer_name**

Fully Normalised Tables

Ⓐ

Order#	Customer#
1	c1
2	c2
3	c3

Order#	Customer_name
1	Smith
2	Jones
3	Smith

Ⓑ

Order#	Customer#
1	c1
2	c2
3	c3

Customer#	Customer_name
c1	Smith
c2	Jones
c3	Smith

Fully Normalised Tables

- Basic error made in previous slide to avoid: associate a determinant (order#) in customer_name table with the transitively dependent attribute customer_name

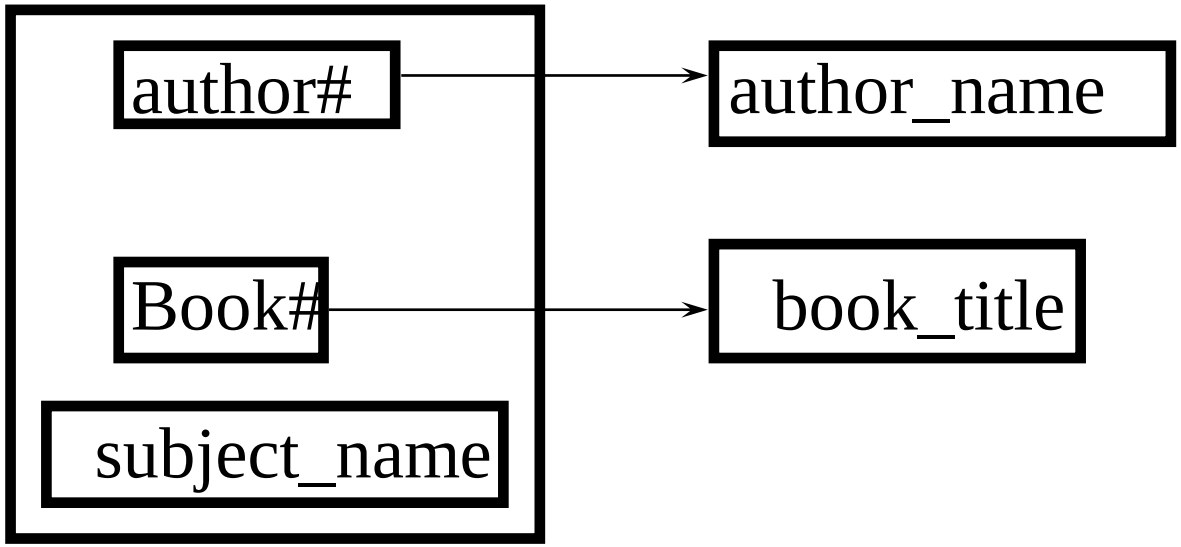
Multivalued Determinacy

Example: Enterprise Rules for Simple Library DB :

A database storing books has the following rules:

- each book has a unique **book#**
- each author has a unique **author#**
- every author has a **name** and every book has a **title**
- each subject classification has a unique **subject_name**
- book# does not distinguish an individual *copy of a book*, only an individual work
- A book may be written by several authors and be classified under several subject_names
- an author may write several books
- a subject_name may be applied to several books

Multi-valued Determinacy



Well normalised tables:

Author (author#, author_name) ✓

Book (book#, book_title) ✓

Author_Book_Subject(author#,book#,subject_name) ✗

Book example

- **Book# B15** is jointly authored by **A2** and **A5** and is classified under subject names **biology** & **physics**.
- If every **author** of a given book is always associated with all the **subject-names** under which the book is classified, then the attribute subject-name can contain certain redundant values

Author#	Book#	Subject-name
A2	B15	biology
A2	B15	physics
A5	B15	biology
A5	B15	physics
A2	B18	physics

Book example

- If subject names biology & physics were deleted from rows 1 and 2, it would be possible to deduce those values from row 3 and 4

Therefore
author-book-subject is well but not fully normalised

Author-Book-Subject

Author#	Book#	Subject-name
A2	B15	biology
A2	B15	physics
A5	B15	biology
A5	B15	physics
A2	B18	physics

Book example

Author#	Book#	Subject-name
A2	B15	biology
A2	B15	physics
A5	B15	biology
A5	B15	physics
A2	B18	physics

- The table is not fully normalised because the same set of subject-names is associated with each author of the same book

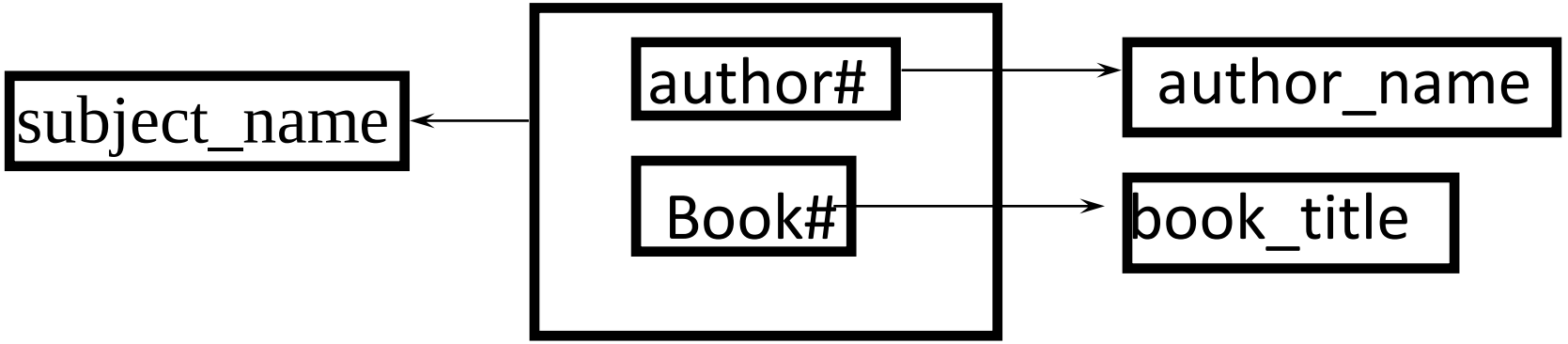
  Book# is said to *multi-determine* author# & subject_name

Note

This would not be true if a different rule were assumed. i.e. that subject-name refers to a subject area within a book for which an individual author was responsible

e.g. delete biology from row 3 => you cannot deduce the info. from elsewhere in the table

Multi-valued Determinacy



If author is responsible for particular subject content of book

Author_book_subject2(author#, book#, subject_name)

Nomalised Book Example

- Full normalisation can be achieved by splitting the table into two parts:

Author#	Book#	Subject-name
A2	B15	biology
A2	B15	physics
A5	B15	biology
A5	B15	physics
A2	B18	physics

Author#	Book#
A2	B15
A5	B15
A2	B18

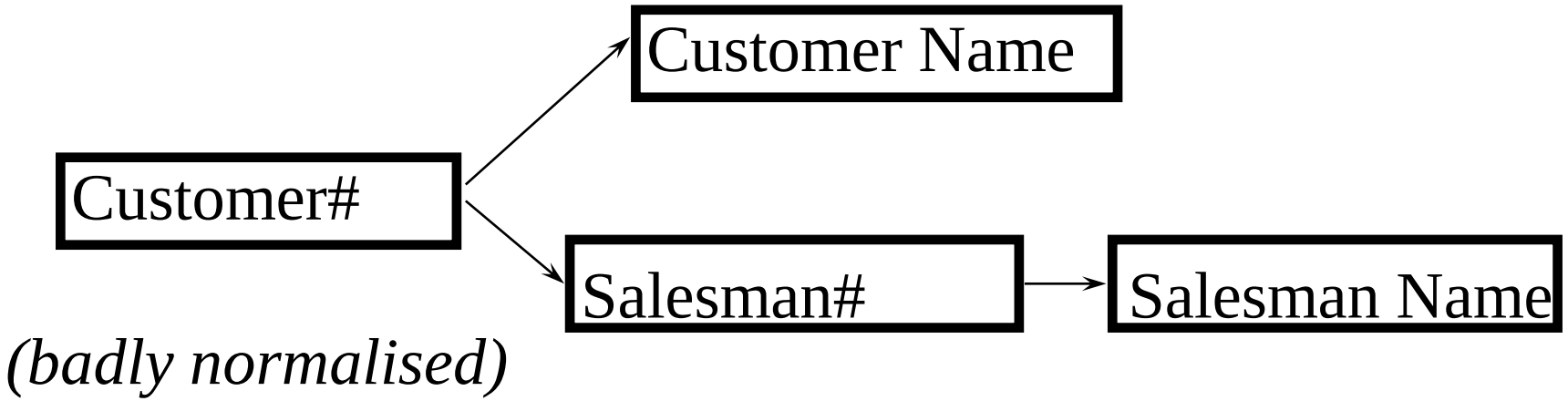
Book#	Subject-name
B15	Biology
B15	Physics
B18	Physics

Advantages of Full Normalisation

Advantages of Full Normalisation

- So far emphasis has been placed on eliminating redundancy
- Further benefits relate to deletion , insertion operations

<u>Deletion Side Effect</u>



(badly normalised)

Advantages of Normalisation(2)

C#	cname	S#	sname
C1	Brown	S4	Jones
C2	Carter	S7	Samson
C3	Cross	S4	Jones
C4	Barns	S8	Baker

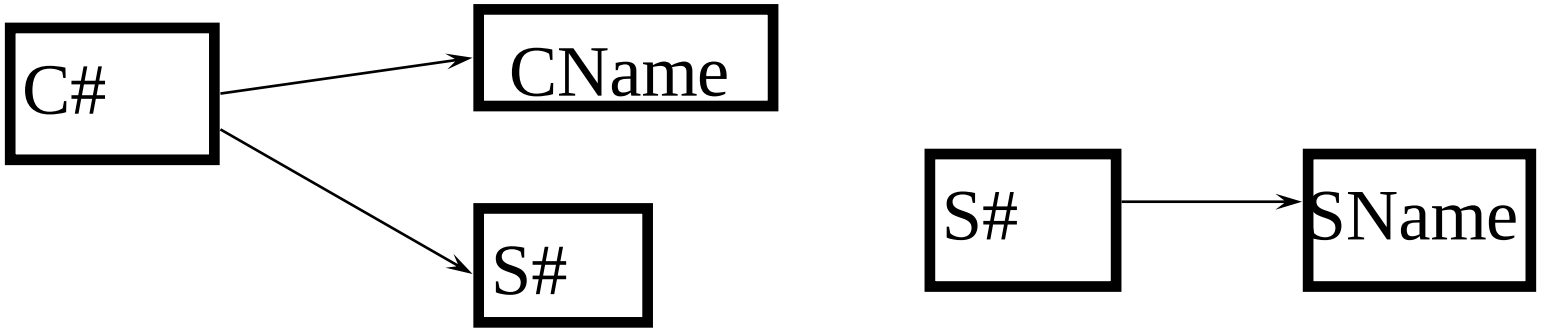
- Delete C2
  => delete whole tuple
  => lose salesman information
- Deleting C# on its own is not allowed as it is an identifier and cannot be null

Advantages of Normalisation(3)

Insertion Side Effect

- Add Salesman S3 whose name is Hall
- You cannot do this until that salesman is associated with a customer, otherwise identifier C# will be null

Should be modelled as:



Advantages of Normalisation(3)

Insertion Side Effect

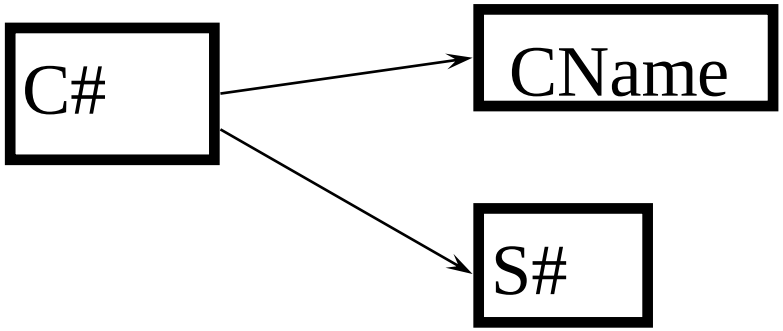
- Add Salesman S3 whose name is Hall
- You cannot do this until that salesman is associated with a customer, otherwise identifier C# will be null

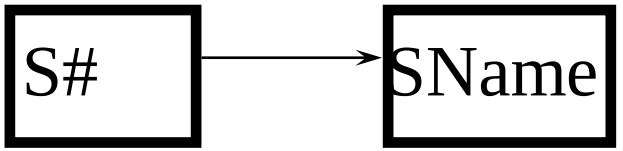
C#	cname	S#	sname
C1	Brown	S4	Jones
C2	Carter	S7	Samson
C3	Cross	S4	Jones
C4	Barns	S8	Baker
??	??	S3	Hall

Advantages of Normalisation(3)

C#	cname	S#	sname
C1	Brown	S4	Jones
C2	Carter	S7	Samson
C3	Cross	S4	Jones
C4	Barns	S8	Baker
??	??	S3	Hall

Should be modelled as:





Advantages of Normalisation(3)

C#	cname	S#	sname
C1	Brown	S4	Jones
C2	Carter	S7	Samson
C3	Cross	S4	Jones
C4	Barns	S8	Baker
??	??	S3	Hall

Should be modelled as:

C#	cname	S#
C1	Brown	S4
C2	Carter	S7
C3	Cross	S4
C4	Barns	S8

S#	sname
S3	Hall
S4	Jones
S7	Samson
S8	Baker