5. Relational Database Design

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 appropiate combination of attribute values

A table conforming to these restrictions is called a normalised table

Exercise

Suggest a relation structure for a company wishing to manage its sales/customer records.

Each customer has a name and is serviced by one salesman. The structure must contain customer numbers, salesman numbers, customer names and salesman names.

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 (2)

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 <u>NOT</u> redundant!

Duplicated vs. Redundant (3)

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	……

=> duplicated data was redundant

Eliminating Redundancy

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

- Preferable to split table into 2 tables

Part#	Part Desc.
P1	bolt
P6	bolt
P4	nut

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

Eliminating Redundancy (2)

- 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 <u>not entirely accurate</u> since attribute values are subject to insertion / change / deletion

Eliminating Redundancy (3)

SP

S#	P#	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:

  " An attribute must only have 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 (2)

<u>Problems</u>:

1. Table is asymmetric representation of symmetrical data
2. Rows can be sorted into s# but not into p#
3. Rows are different length 'cos of 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
S7	Edwards	P8
		P2
		P6

→

S#	SName	P#
S5	Wells	P1
S2	Heath	P1
S2	Heath	P4
S7	Edwards	P8
S7	Edwards	P2
S7	Edwards	P6

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

Elimination of Repeating Groups(2)

<u>Alternate method:</u>

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

Conclusion

- Its 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?

Codd's Normal Forms

- Codd identified some rules which govern the way we create tables so as to avoid anomolies 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 olnly a single value from that domain
- 2nd Normal Form
  - A relation is in 2nd normal form if, in addition to satifying 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.	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).

SO WHAT IS FUNCTIONAL DEPENDENCY?

Determinants

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

Note: A is determinant of B can be written as

$$A \longrightarrow B$$

<u>**Note:**</u> In the special case where duplicate values of A are not allowed in a table (i.e. A is a key) then A is obviously a determinant on B

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.

$$p\# \longrightarrow p\_desc$$

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

$$p\# \longrightarrow Qty\_in\_Stock$$

Determinants (3)

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 ?

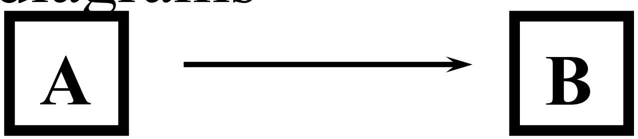
Superfluous Attribute

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

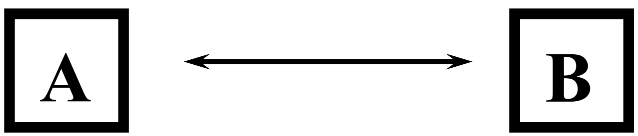
- We assume determinants do not contain any superfluous attributes

Determinancy Diagrams

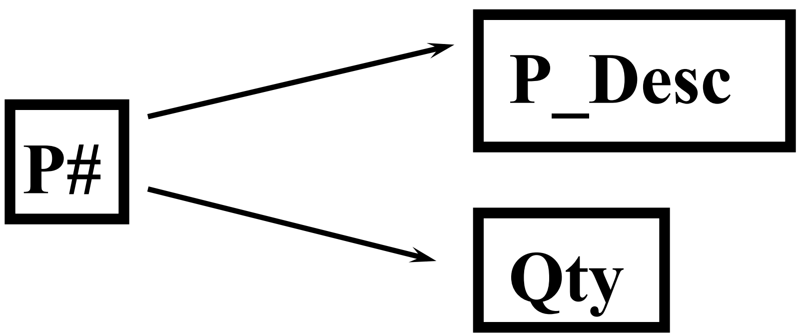
- We need a notation to express where one attribute determines another => we use determinancy diagrams



A is determinant of B



A is determinant of B and vice versa



Determinancy diagram of stock table

Determinancy Diagrams (2)

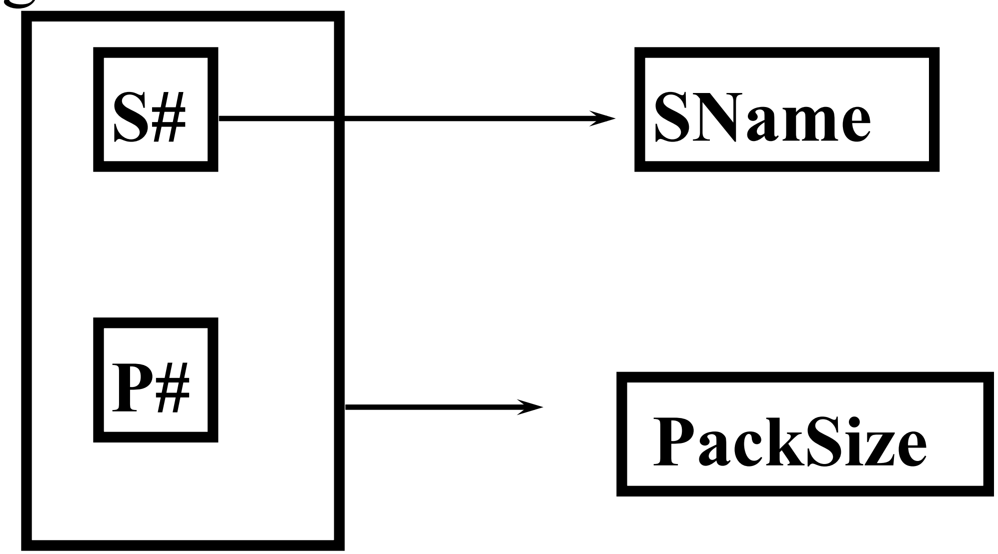
EXAMPLE

- 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 supplier may supply many different parts in many different pack sizes
- A part may be supplied by many different suppliers in many different pack sizes
- A given supplier supplies a given part in just one pack size

Determinancy Diagrams (3)

Supp-Part (S#, Sname, P#, PackSize)

- Can recognise determinants by drawing determinancy diagrams

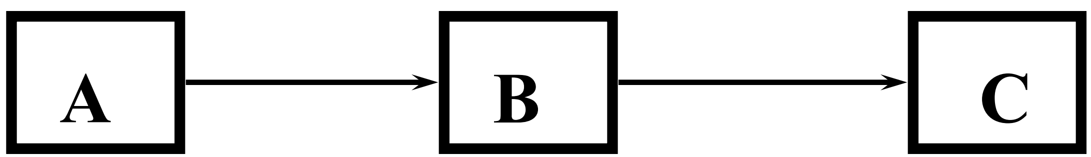


{ S#, P# } is determinant for packsize

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 (2)

Example:

Employee (Employee#, Employee_name, Salary)

Rules:

- No two rows should have the same value for Employee#

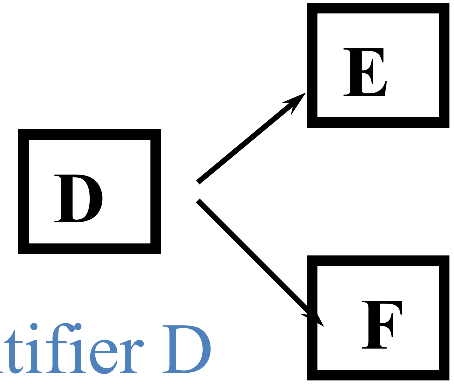
=> Employee# is a row identifier of the table

- Where a composed attribute forms the identifier

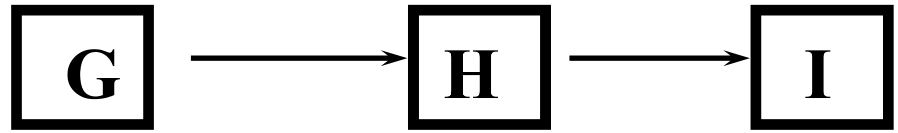
=> no component part (of identifier) can be null (entity constraint)

Identifiers (3)

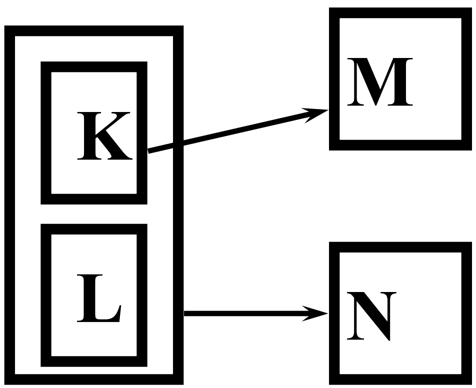
Examples:



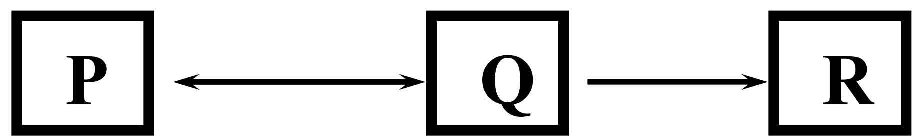
Identifier D



Identifier G



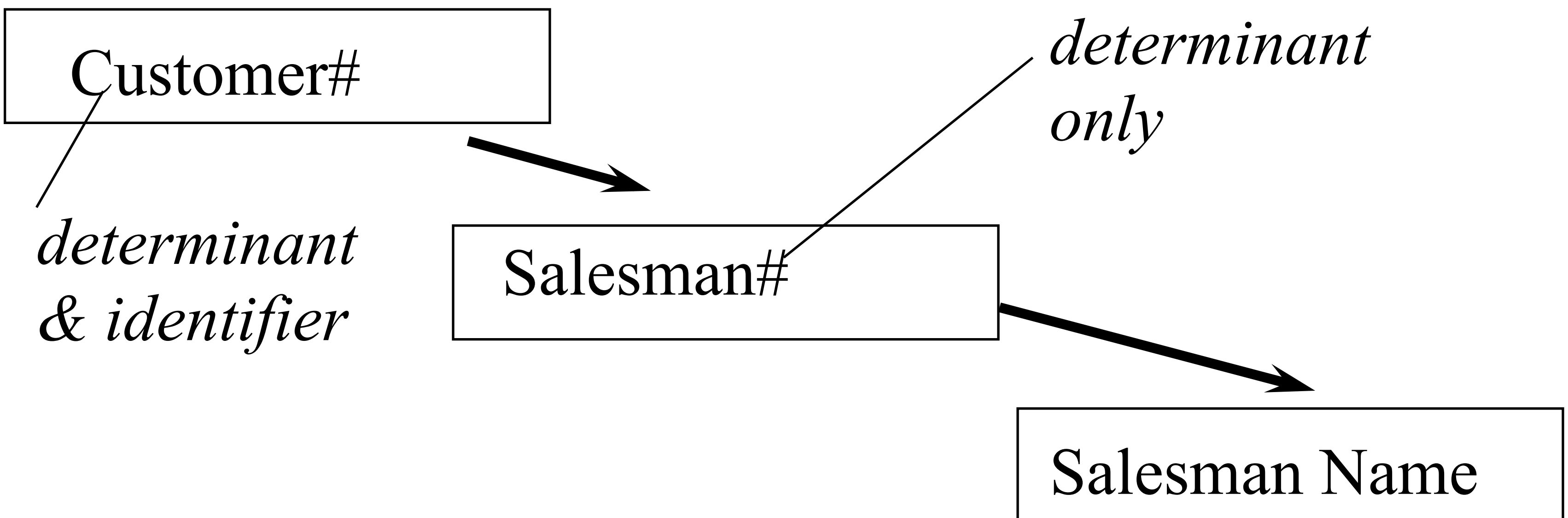
Identifier {K, L}



Identifier P & Q
2 candidate identifiers

Determinancy & Redundancy

- Given a determinancy diagram (developed from enterprise rules) we can detect and eliminate table structures which could contain redundant data



Determinancy & Redundancy(2)

- 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# values will be associated with the same salesman name
  => table can contain redundant values of salesman

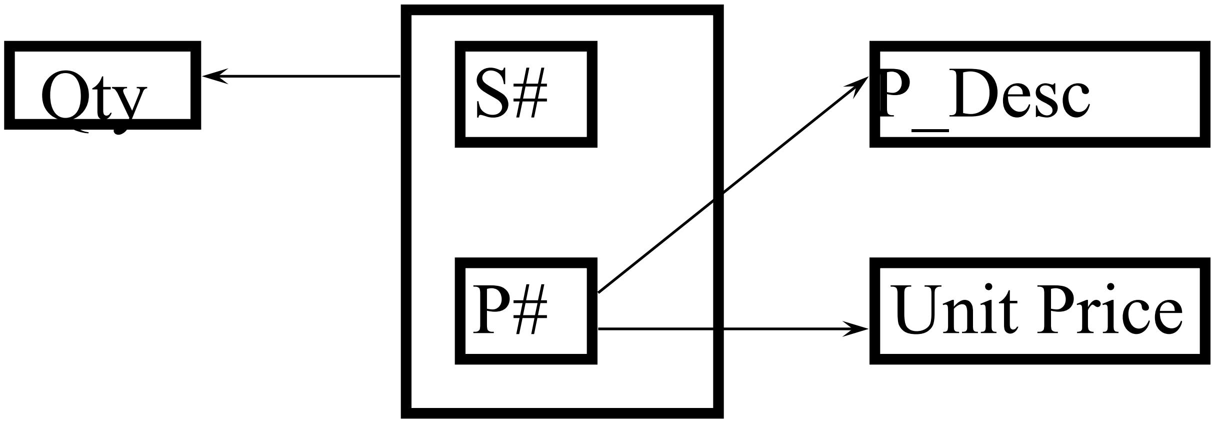
Determinancy & Redundancy (3)

- 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

Determinancy & Redundancy (4)

Example 2



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

Transforming tables in *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"

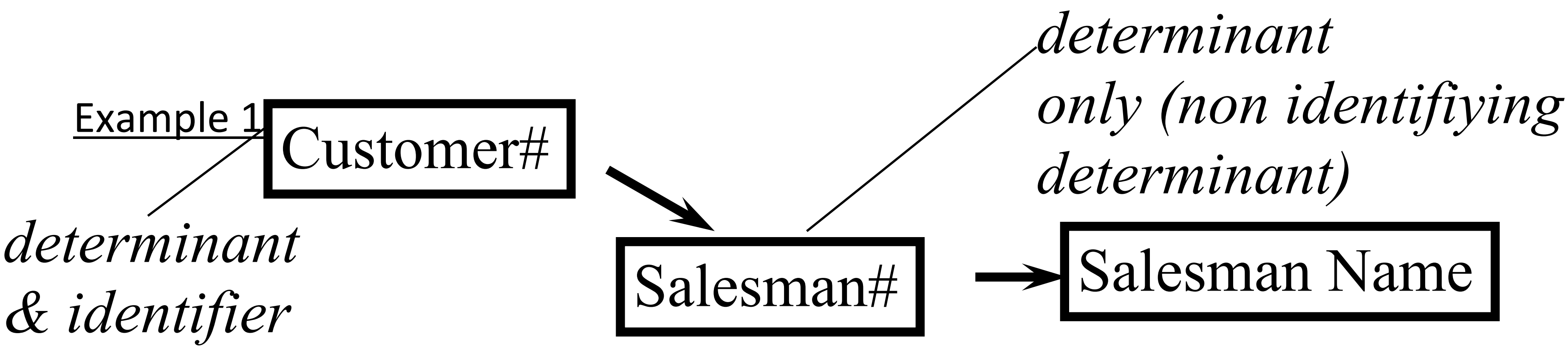
Transformation into Well N.F.

- A determinant which is not a candidate identifier is called a non identifying determinant

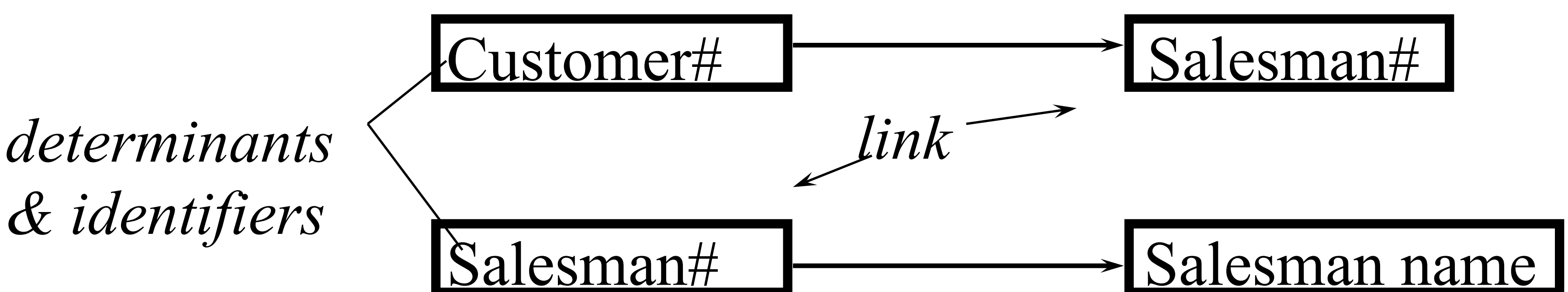
- To transform a badly normalised (non BCNF) table into well normalised tables:

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

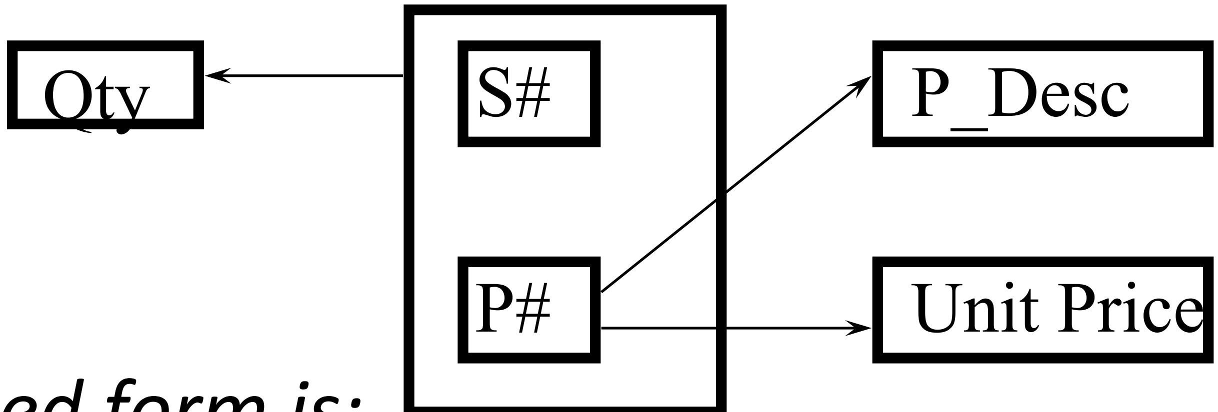


in normalised form is:

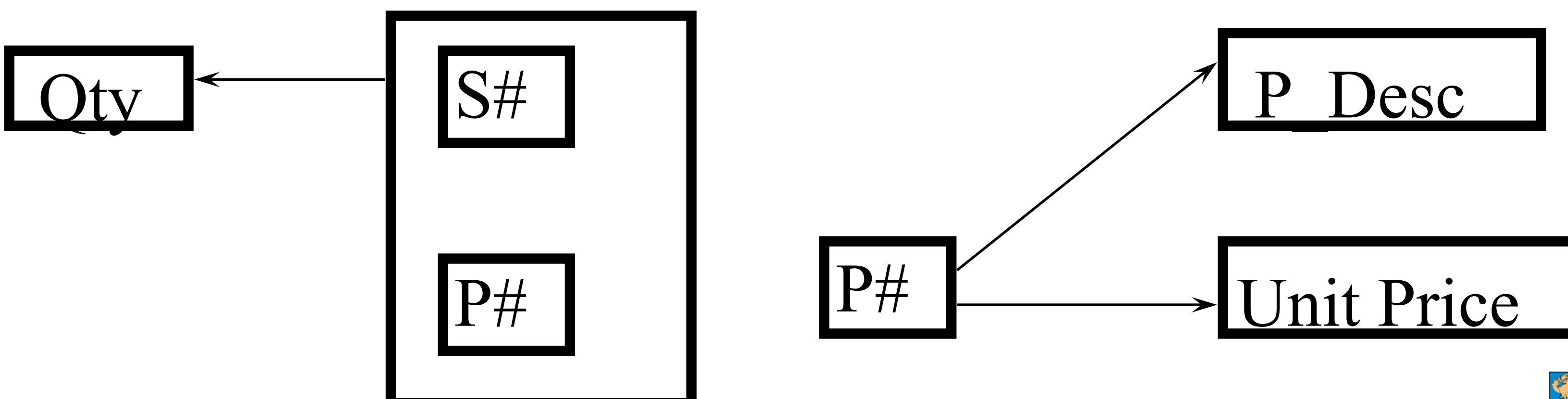


Example of BCNF Normalisation (2)

Example2



in normalised form is:



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 desireable.

Fully Normalised Tables (2)

<u>Badly Normalised (hidden transitive dependency)</u>

order# → customer# → customer name

<u>Normalised</u>

A

order# → customer#

order# → customer name

B

order# → customer#

customer# → customer name

Fully Normalised Tables (3)

A contains redundant data e.g.

order#	customer#
1	C1
2	C2
3	C1

order#	customer name
1	Smith
2	Jones
3	Smith

- If delete smith from row 1 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 (order# 3 or 4)
  - uses order# to search customer name table for corresponding customer name

Fully Normalised Tables (3)

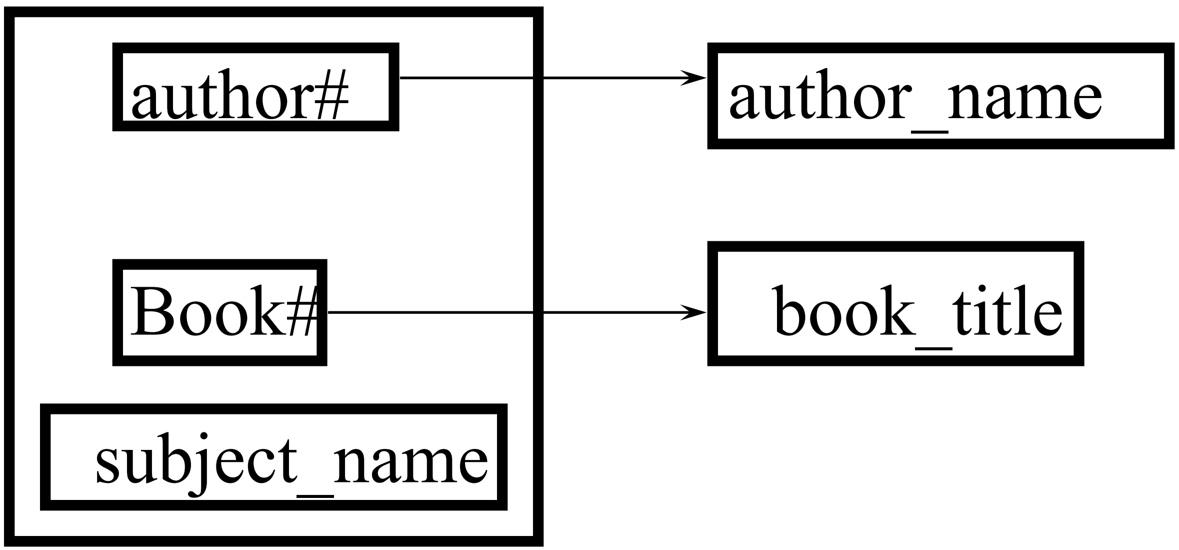
- Basic error: associate in customer name table a determinant (order#) with the transitively dependant attribute customer name knowledge and data engineering group

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(1)



Well normalised tables:

Author (author#, author_name)

Book (book#, book_title)

Author_Book_Subject(author#,book#,subject_name)

Book example (2)

- 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.

Book example (3)

- 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 <u>well but</u> <u>not fully</u> <u>normalised</u>

Author-Book-Subject

author#	book#	subject-name
A2	B15	biology
A2	B15	physics
A5	B15	biology
A5	B15	physics
A2	B18	physics

Book example (4)

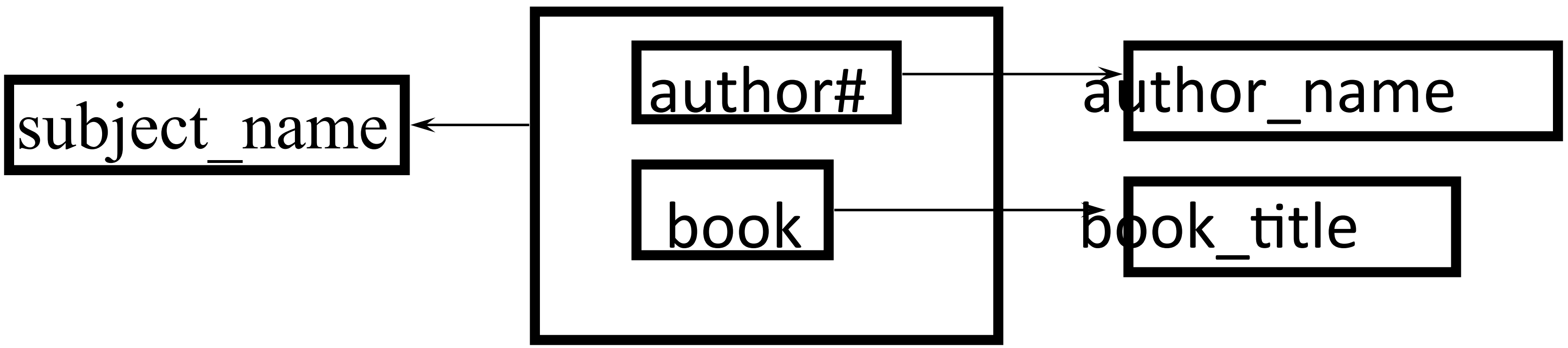
- The table is not fully normalised because the same set of subject-names is associated with each author of the same book.
  Book# is said to *multi-determine* author# & subject_name.

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.

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

Multi-valued Determinacy(2)



If author is responsible for particular subject content of book

Author_book_subject2(author#, book#, subject_name) knowledge and data engineering group

Nomalised Library Example

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

Author-Book

author#	book#
A2	B15
A5	B15
A2	B18

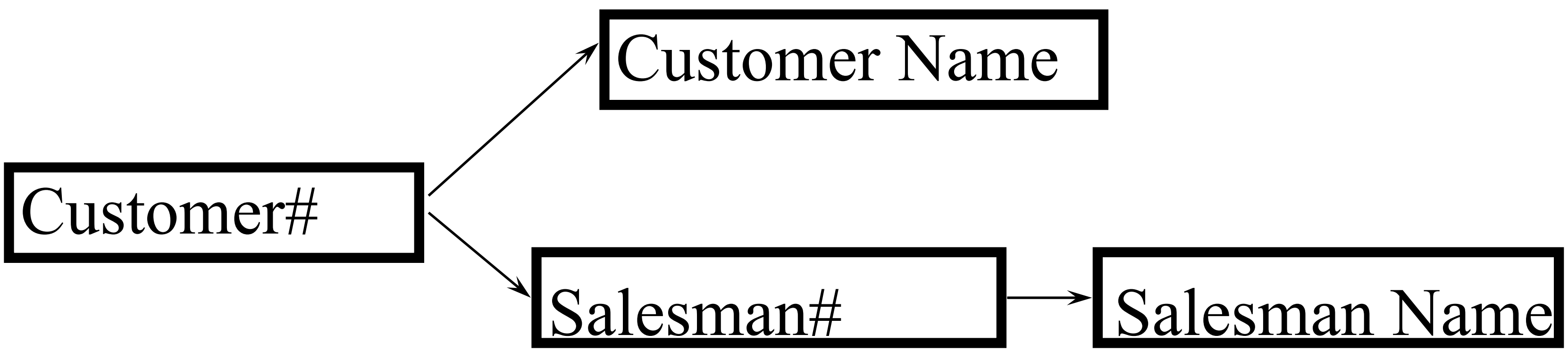
Book-Subject

book#	subject-name
B15	biology
B15	physics
B18	physics

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 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: