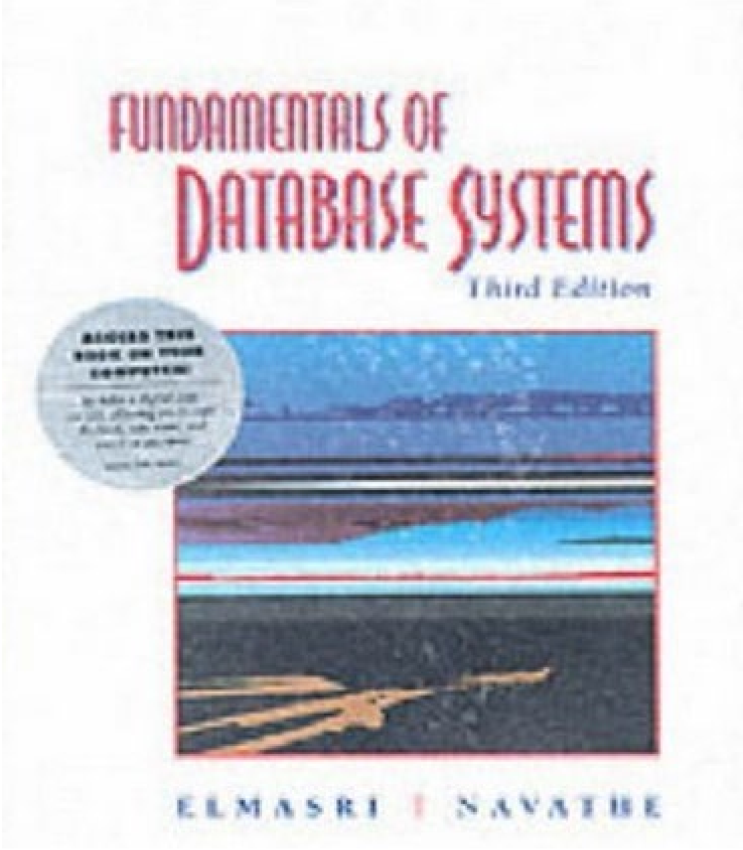
TOPIC: *NORMALISATION PART 2*

C230 Database Systems

FUNDAMENTALS OF DATABASE SYSTEMS
ELMASRI AND NAVATHE BOOK

See Chapter 14

(in 3rd Edition)



DEFINITION:
Functional Dependency

Functional dependency is one of the main concepts associated with normalisation and describes the *relationship between attributes.*

If A and B are attributes of a relation R, then **B is functionally dependent (FD) on A** if each value of A is associated with exactly one value of B.

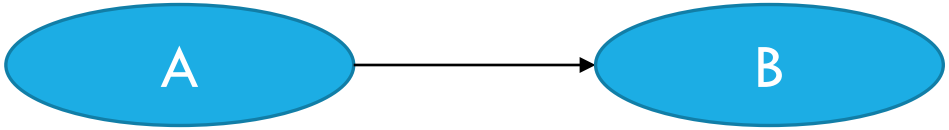
i.e., values in B are uniquely determined by values of A

TERMINOLOGY:
FUNCTIONAL DEPENDENCY (FD)

A $\longrightarrow$ B :

FD from A to B

B is FD on A



NOTES ON NOTATION:

$A \longrightarrow B$ does not necessarily imply $B \longrightarrow A$

$A \leftrightarrow B$ denotes $A \longrightarrow B$ and $B \longrightarrow A$

$A \longrightarrow \{B, C\}$ denotes $A \longrightarrow B$ and $A \longrightarrow C$

$\{A, B\} \longrightarrow C$ denotes that it is the **combination** of A and B that uniquely determines C.

TERMINOLOGY: CANDIDATE KEY (CK)

Every relation has one or more candidate keys. A candidate key (CK) is one or more attribute(s) in a relation with which you can determine all the attributes in the relation.

Recall we pick one such candidate key as the primary key of a relation.

EXAMPLE 3: FINDING THE FUNCTIONAL DEPENDENCIES – GIVEN THE PRIMARY KEY

For the company schema, consider the following alternative schema to hold information on employees and projects:

`emp_proj(ssn, pnumber, hours, ename, pname, plocation)`

What are the functional dependencies?

- Think of this question as … "which attribute can be uniquely determined from another attribute"
- Begin with any known PK or CK

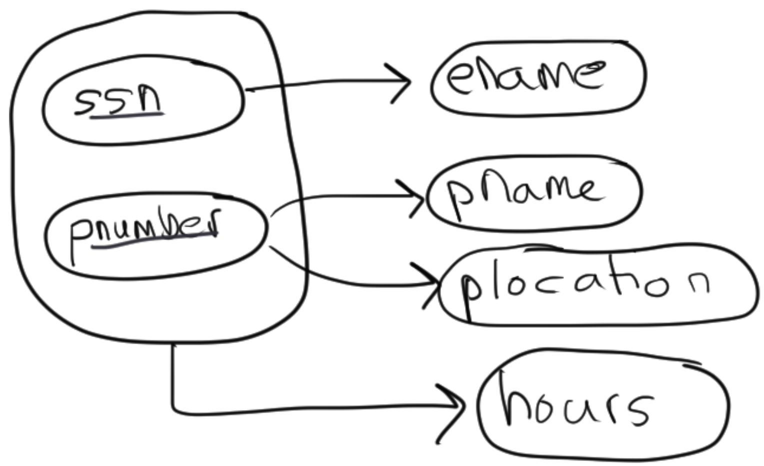
Can represent these FDs graphically:

`emp_proj(ssn, pnumber, hours, ename, pname, plocation)`

ssn ⟶ ename

pnumber ⟶ {pname, plocation}

{ssn, pnumber} ⟶ hours



IMPORTANT TO NOTE:

A functional dependency is a property of a relation schema *R* and cannot be inferred automatically but instead must be defined explicitly by someone who knows the **semantics** of *R*

i.e.

You will either be:

- explicitly given all FDs.
- given enough information about the attributes and the domain to *reasonably* infer the FDs (perhaps having to make certain assumptions).

TYPES OF FUNCTIONAL DEPENDENCIES

1. Full Functional Dependency:

A functional dependency $\{X,Y\} \rightarrow Z$ is a **full** functional dependency if when some attribute (either X or Y) is removed from the LHS the dependency **does not hold**.

Note: There may be any number of attributes on LHS

2. Partial Functional Dependency:

A functional dependency $\{X,Y\} \rightarrow Z$ is a **partial** functional dependency if some attribute (either X or Y) can be removed from the LHS and the dependency **still holds.**

Note: There may be any number of attributes on LHS

CONSIDER EXAMPLE 3 AGAIN:

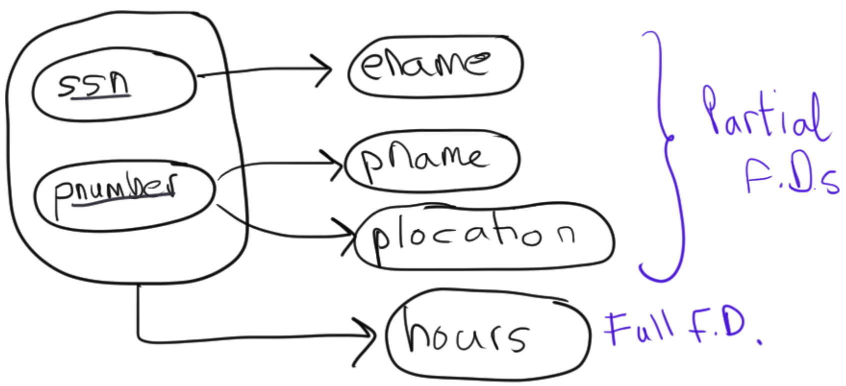
`emp_proj(ssn, pnumber, hours, ename, pname, plocation)`

Are the following Full or Partial Functional Dependencies?

See menti.com

{ssn, pnumber} → hours

{ssn, pnumber} → ename



TYPES OF FUNCTIONAL DEPENDENCIES

3. Transitive Dependency:

A functional dependency **X** → **Y** is a transitive dependency in the table/relation R if there is a set of attributes **Z** that is neither a candidate key nor a subset of any key of R and both:

X → **Z** and

Z → **Y**

hold.

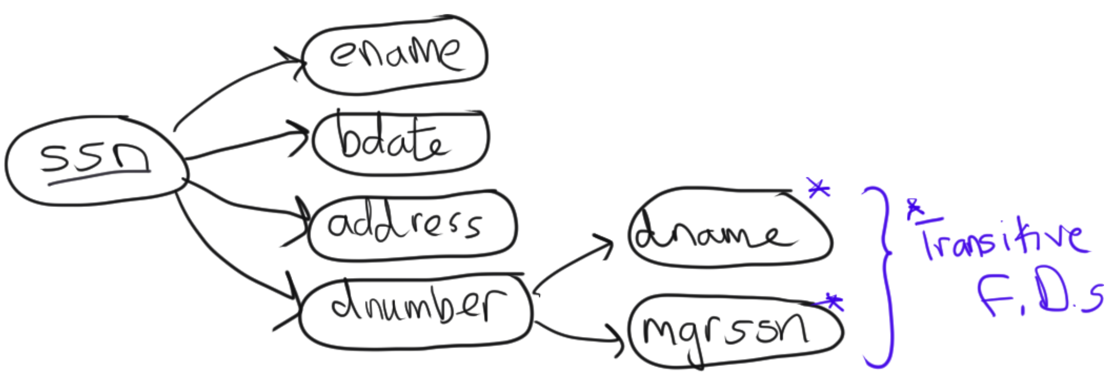
EXAMPLE 4:
Consider information on employees and departments

emp_dept(ename, ssn, bdate, address, dnumber, dname, dmgrssn)

The functional dependencies are:

ssn → {ename, bdate, address, dnumber}

dnumber → {dname, dmgrssn)



EXAMPLE 4:
An example of a transitive dependency

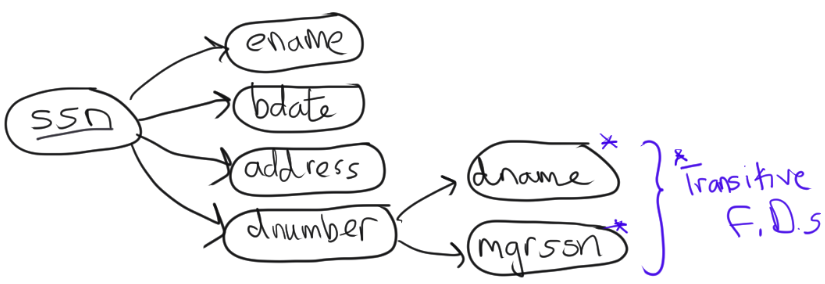
The dependency:

ssn → dmgrssn

is transitive through `dnumber` because both the following hold:

ssn → dnumber

dnumber → dmgrssn



But `dnumber` is neither a key or a subset of the key.

EXAMPLE 5:

Given the following table to hold student data:

`student(id, name, course, assocCollege, courseCoordinator)`

and the following Functional Dependencies:

id $\longrightarrow$ name

id $\longrightarrow$ course

course $\longrightarrow$ assocCollege

course $\longrightarrow$ courseCoordinator

EXAMPLE 5:
What is the candidate key?
What are the full dependencies?
What are the transitive dependencies?

Given the following table to hold student data:

```
student(id, name, course, assocCollege,
courseCoordinator)
```

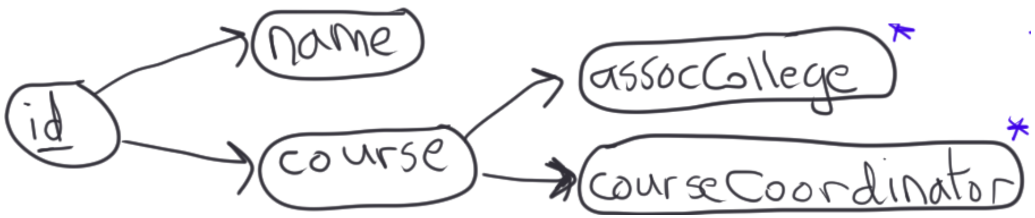
and the following Functional Dependencies:

id → name

id → course

course → assocCollege

course → courseCoordinator



EXAMPLE 6:
Draw the functional dependency diagram and find the candidate key

Consider the table R with 5 attributes

`R(A, B, C, D, E)`

and the following functional dependencies:

$A \rightarrow B$

$B \rightarrow A$

$B \rightarrow C$

$D \rightarrow A$

`R(A, B, C, D, E)`

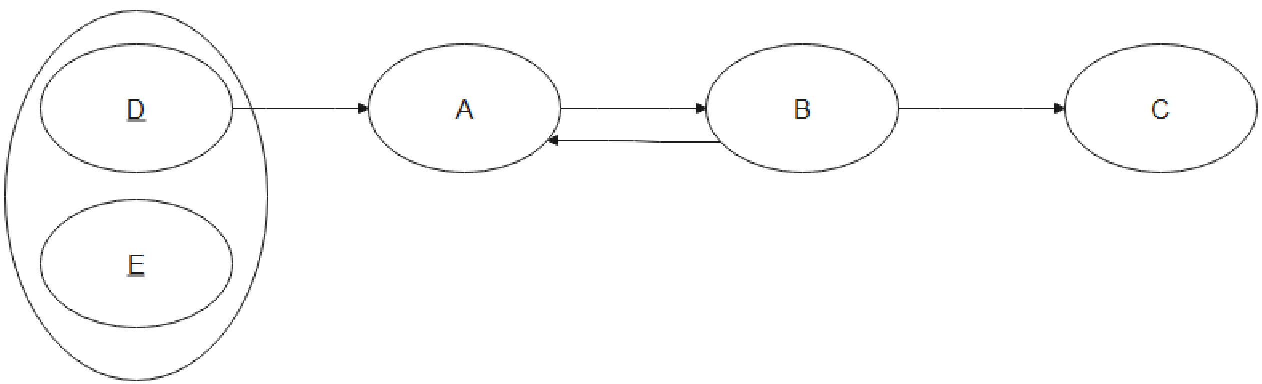
and the following functional dependencies:

A → B

B → A

B → C

D → A



Inference rules for Functional Dependencies

Typically the main **obvious** functional dependencies are specified for a schema

– call these F.

However many others can be inferred from F

– call these closure of F: F^+

FOR EXAMPLE:

$F = \{ \quad A \rightarrow \{B, C, D, E\}$

$\qquad E \rightarrow \{F, G\} \quad \}$

Some other FDs which can be inferred:

$A \rightarrow A$

$A \rightarrow \{F, G\}$

$E \rightarrow F$

etc.

Inference Rules for FDs:

1. Reflexive: Trivially, an attribute, or set of attributes, always determines itself.

2. Augmentation: if $X \rightarrow Y$ can infer $XZ \rightarrow YZ$

3. Transitive: if $X \rightarrow Y$ and $Y \rightarrow Z$ can infer $X \rightarrow Z$

4. Decomposition: if $X \rightarrow YZ$ can infer $X \rightarrow Y$

5. Union (additive): if $X \rightarrow Y$ and $X \rightarrow Z$ can infer if $X \rightarrow YZ$

6. Pseudotransitive: if $X \rightarrow Y$ and $WY \rightarrow Z$ can infer $WX \rightarrow Z$

*Note: Rules 1, 2 and 3 are together called **Armstrongs's Axioms**

IMPORTANT CONCEPTS

Duplicated Data versus Redundant Data

Problems with un-normalised tables and maintaining redundant data

Trade off of un-normalised versus normalised tables

What is functional dependency – how to find it

What are full, partial and transitive dependencies – how to find them

DEFINITION:
FIRST NORMAL FORM (1NF)

A table is in 1NF if it satisfies the following:

The table must not have any **repeating groups**

Repeating groups: a group of attributes that occur a variable number of times in each record (non-atomic)

FIRST NORMAL FORM (1NF)

To ensure first normal form, choose an appropriate primary key (if one is not already specified) and if required, split table in to two or more tables to remove repeating groups

EXAMPLE 7:

Consider information on customers (unique number, name, address and their credit limit) and invoices issued to them (unique invoice number, date of invoice and amount in euros). Note that a customer can have many invoices issued to them.

```
customer(cNo, name, street, city,
credLim, invNo, invDate, amount)
```

Repeating Groups?

First Normal Form?

EXAMPLE 7

`customer(cno, name, street, city, credLim, invno, invDate, amount)`

To ensure 1NF, choose appropriate Primary Key ....

`cNo` **and** `invNo` **as primary key giving:**

`customer(cNo, invNo, name, street, city, credLim, invDate, amount)`

DEFINITION:
SECOND NORMAL FORM (2NF)

A relation in 2NF must be in 1NF and satisfy the following:

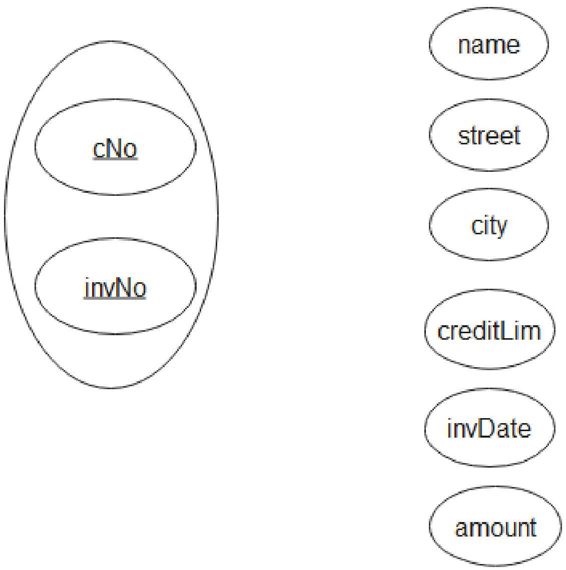
Where there is a composite primary key, all non-key attributes must be dependent on the **entire** primary key.

If partial dependencies exists create new relations to split the attributes such that the partial dependency no longer holds

check for partial dependencies and remove

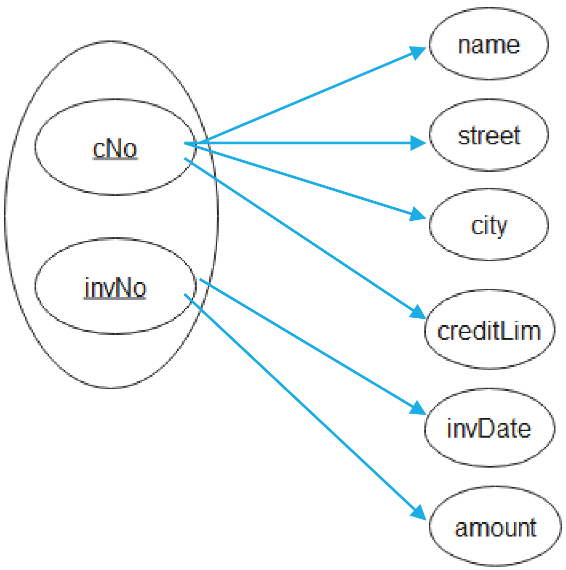
EXAMPLE 7:

customer(cNo, invNo, name, street, city, credLim, invDate, amount)



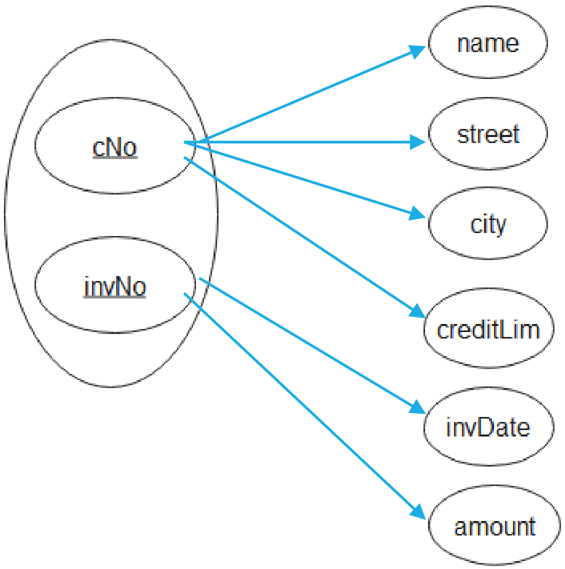
EXAMPLE 7:

customer(<u>cNo, invNo</u>, name, street, city, credLim, invDate, amount)



EXAMPLE 7:

customer(<u>cNo, invNo</u>, name, street, city, credLim, invDate, amount)



customerInvoice(<u>cNo, invNo</u>)

customer(<u>cNo</u>, name, street, city, credLim)

invoice(<u>invNo</u>, invDate, amount)

EXAMPLE 8:

Consider information on products that customers buy (e.g. the contents of their online basket). Information stored on customers is: unique customer number, name and address. The data stored on the products ordered is: unique product number, product description, unit price per product and quantity of each product required by the customer. The schema is:

```
purchase(CNo, ProdNo, cname, street, city, prodDesc,
price, quantity)
```

QUESTIONS:

`purchase(CNo, ProdNo, cname, street, city, prodDesc, price, quantity)`

- Is this table in first normal form?
- Draw a functional dependency diagram
- Is this table in second normal form?
- If not, what problems occur by the table not being in 2NF?
- If not, create a set of tables in 2NF

1NF?

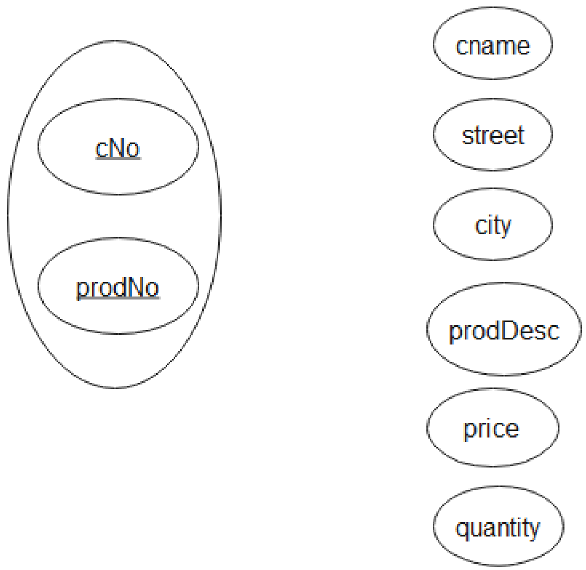
`purchase(CNo, ProdNo, cname, street, city, prodDesc, price, quantity)`

No primary key so not in 1NF.

A suitable primary key (using existing attributes) is a composite key of `CNo` and `ProdNo`

Draw the Functional Dependencies:

purchase(CNo, prodNo, cname, street, city, prodDesc, price, quantity)



Problems caused by `purchase` table not being in 2NF:

`purchase(cNo, prodNo, cname, street, city, prodDesc, price, quantity)`

Duplication of data:

- Every time a product is purchased by a customer the customer name, street etc. is stored again
- Every time a product is purchased, its description and price is stored again.

Create a set of tables in 2NF

Removing the partial dependencies means:

- Attributes that are partially dependent on the PK should move to a new table;
- The attribute on which they were dependent should be the PK of the new table but this attribute should not be removed from the original table

Giving the tables:

`purchase(cNo, prodNo, quantity)`

`customer(cNo, cname, street, city)`

`product(prodNo, prodDesc, price)`

N.B. Make sure each table has its own PK

DEFINITION:
THIRD NORMAL FORM (3NF)

A relation is in 3NF if it is in 2NF and there are no dependencies between attributes that are not primary keys. That is, no transitive dependencies exist in the table.

EXAMPLE 8 *extended*:

Consider the following information stored per product: unique product number (PK), product description and unit price and the number of the product in stock; also stored is the unique ID of the supplier of the product, and the supplier's details: name and address details:

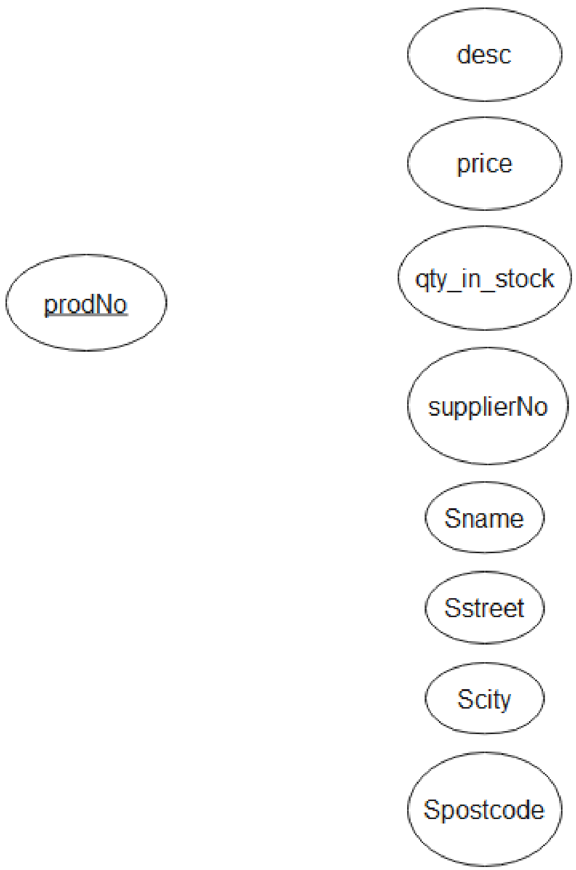
```
product(prodNo, desc, price,
qty_in_stock, supplierNo, Sname,
Sstreet, Scity, SPostcode)
```

QUESTIONS:
EXAMPLE 8 extended

```
product(prodNo, desc, price,
 qty_in_stock, supplierNo,Sname,
 Sstreet, Scity, SPostcode)
```

- Is this table in first normal form?
- Draw a functional dependency diagram
- Is this table in second and third normal form?
- If not, create a set of tables in 3NF

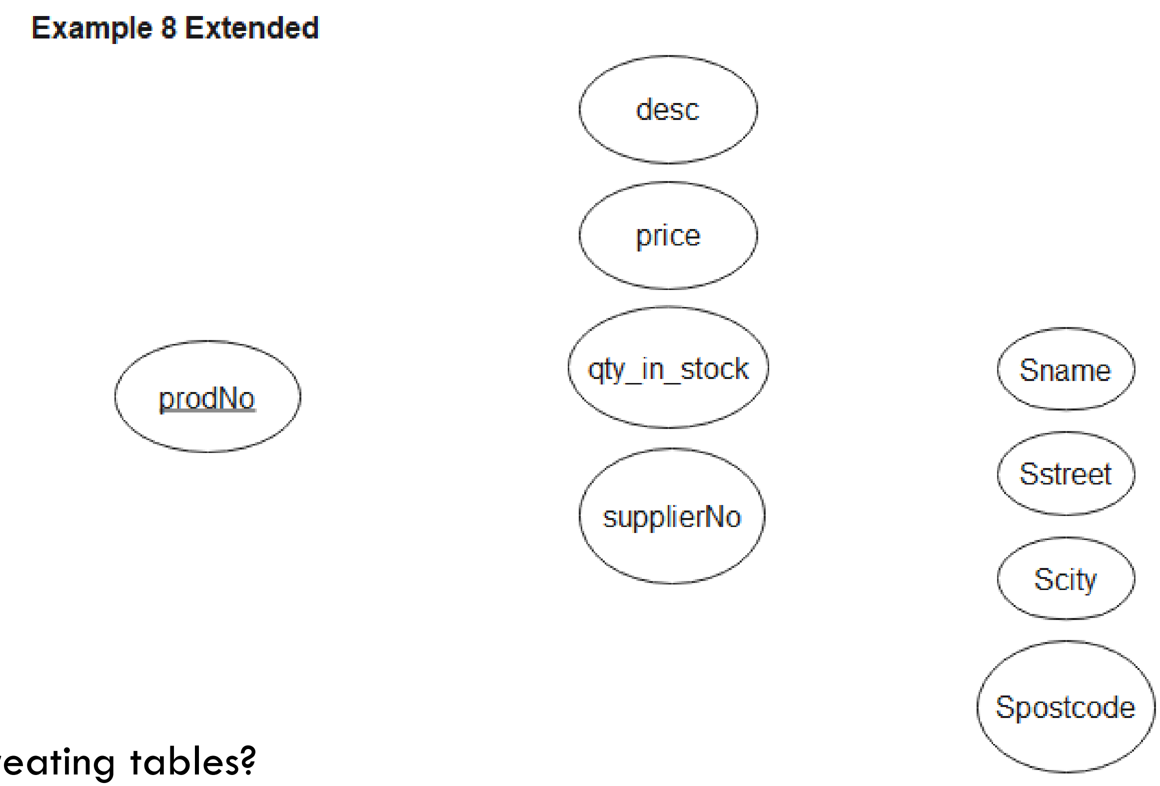
DEPENDENCY DIAGRAM FOR EXAMPLE 8 EXTENDED



Creating tables?

prodNo, desc, price, qty_in_stock, supplierNo, Sname, Sstreet, Scity, SPostcode

DEPENDENCY DIAGRAM FOR EXAMPLE 8 EXTENDED



Note: how we are creating links between the tables with Foreign Keys

Creating tables?

`product(prodNo, desc, price, qty_in_stock, supplierNo)`

`supplier(supplierNo, Sname, Sstreet, Scity, Spostcode)`

BOYCE-CODD NORMAL FORM (BCNF)

Only in rare cases does a 3NF table not meet the requirements of BCNF.

These cases are when a table has more than one candidate key - depending on the functional dependencies, a 3NF table with two or more overlapping candidate keys may or may not be in BCNF.

If a table in 3NF **does not** have multiple overlapping candidate keys then it is guaranteed to be in BCNF

SUMMARY: Steps to normalise to 3NF

- Identify appropriate Primary Key if not already given (this puts table in to 1NF)
- Draw diagram of Functional Dependencies from the primary key.
- Identify if dependencies are Full, Partial or Transitive.
- Using diagram of functional dependencies from previous step:
  - Normalise to 2NF by removing partial dependencies – creating new tables as a result. <u>Ensure all new tables have Primary Keys</u>
  - Normalise to 3NF by removing transitive dependencies (if they exist), creating new tables as a result. <u>Ensure any new tables have Primary Keys and are in 2NF</u>
  - Check that all resulting tables are themselves in 1NF, 2NF and 3NF (in particular, make sure they all have PKs of their own)

EXAMPLE 9:

An un-normalised staff relation has the following structure and description (next slide):

```
staff(sNo, sName, sAddress, deptNo,
deptName, managerNo, skilliD, skillName,
sCourseDate, sCourseDuration)
```

9.1. Where does duplication result from this relation design?

9.2. What is a suitable Primary Key to ensure the staff table is in 1NF?

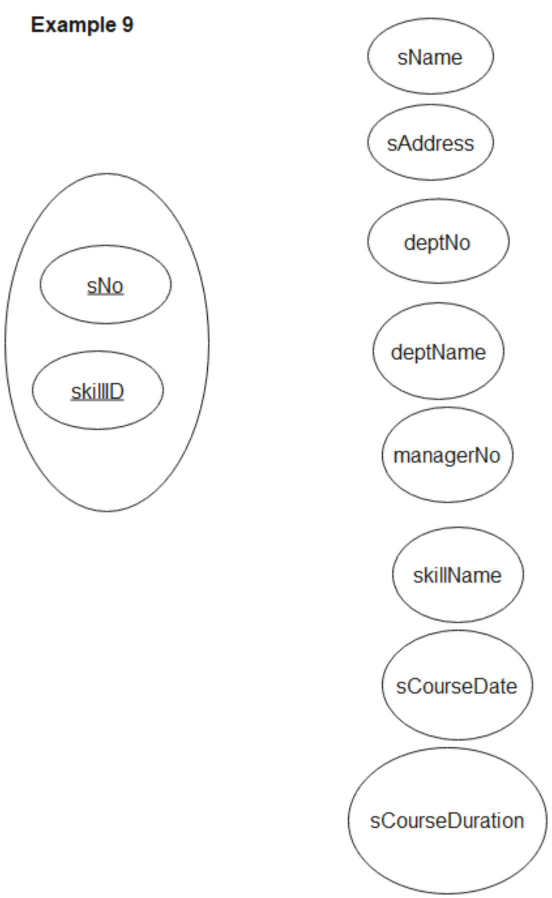
9.3. What attributes are fully functional dependent on the Primary Key?

Description 9(a):

```
staff(sNo, sName, sAddress, deptNo, deptName,
managerNo, skilliD, skillName, sCourseDate,
sCourseDuration)
```

A staff member has an associated number (sNo, which is unique for each staff member), a name and an address and works in a particular department. Each department has a number (unique), name and manager. A department has many staff but a staff member can only work for one department. A staff member can undertake a number of courses to gain new skills for their job. skilliD uniquely identifies the skill, which has also a name (skillName). **For each skill, courses are offered on a regular basis and staff can take the course at a date that suits them and complete the course at their own pace.** sCourseDate describes the date when a staff member undertakes the course for a particular skill and sCourseDuration describes the time that the staff member took to complete the course. A staff member cannot undertake more than one course to acquire a new skill.

FUNCTIONAL DEPENDENCIES



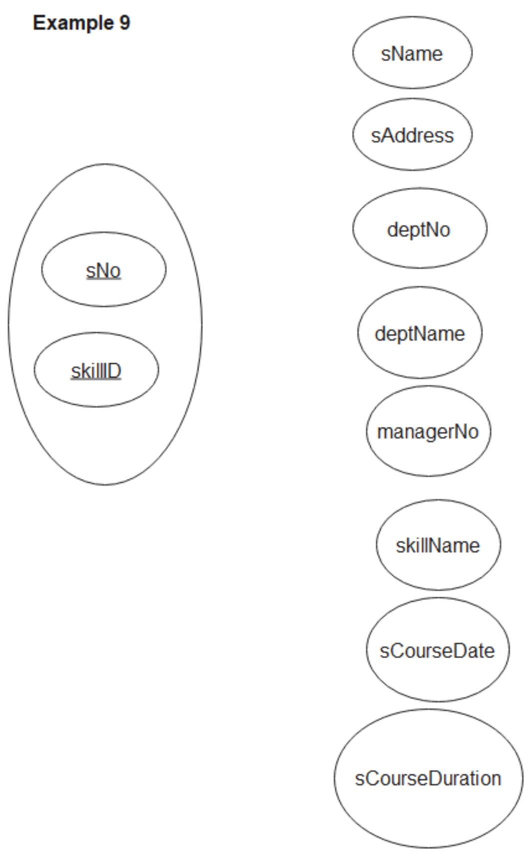
For each skill, courses are offered on a regular basis and staff can take the course at a date that suits them and complete the course at their own pace

Description 9(b)

```
staff(sNo, sName, sAddress, deptNo,
deptName, managerNo, skilliD, skillName,
sCourseDate, sCourseDuration)
```

A staff member has an associated number (sNo, which is unique for each staff member), a name and an address and works in a particular department. Each department has a number (unique), name and manager. A department has many staff but a staff member can only work for one department. A staff member can undertake a number of courses to gain new skills for their job. skilliD uniquely identifies the skill, which has also a name (skillName). **For each skill, courses are offered once at a certain date and for a certain duration and staff must take the course on that date:** sCourseDate describes the date of the course; sCourseDuration describes the length (in days) of the course. A staff member can undertake as many different courses as they wish.

FUNCTIONAL DEPENDENCIES



For each skill, courses are offered once at a certain date and for a certain duration and staff must take the course on that date

EXAMPLE 10: Winter 2019 Exam Paper question on Normalisation

A courier company keeps track of packages that are to be delivered to recipients, by couriers, in the following table:

```
courier(packageID, recipientCode, recipientName,
recipientAddr, recipientMobile, instructions, dateRec,
dateDelivered, courierID, cName, cMobile)
```

Stored in the `courier` table are: a unique package id (`packageID`) which is the primary key of the table, a code (`recipientCode`) which is unique to each recipient, and the name, address and mobile number of the recipient of the package (`recipientName`, `recipientAddr` and `recipientMobile`), delivery instructions (`instructions`), the date the package was received by the courier (`dateRec`), the date the courier delivers the package (`dateDelivered`), and details of the courier who delivers the package: an ID (`courierID`) which is unique to each courier, in addition to the courier's name (`cName`) and phone number (`cMobile`).

```
courier(packageID, recipientcode,
recipientname, recipientaddr, recipientmobile,
instructions, daterec, datedelivered,
courierid, cname, cmobile)
```

(i) By using the primary key given in the `courier` table, draw a functional dependency diagram showing the functional dependencies between all attributes and the key attribute. Clearly indicate on the diagram any full, partial or transitive dependencies and state any assumptions made. *(8)*

(ii) Normalise the `courier` table to third normal form, explaining the steps involved at each stage. *(8)*