Introduction

It should be a requirement of the job that business analysts document process AND data requirements…

Process create, read, update and delete data – they manipulate data. Process that aren't manipulating data aren't doing anything. Processes exist to manipulate data and process simply cannot operate without the right data to manipulate at the right points in the process. The following diagram illustrates how processes are the pivot around which solutions are specified, designed and implemented:
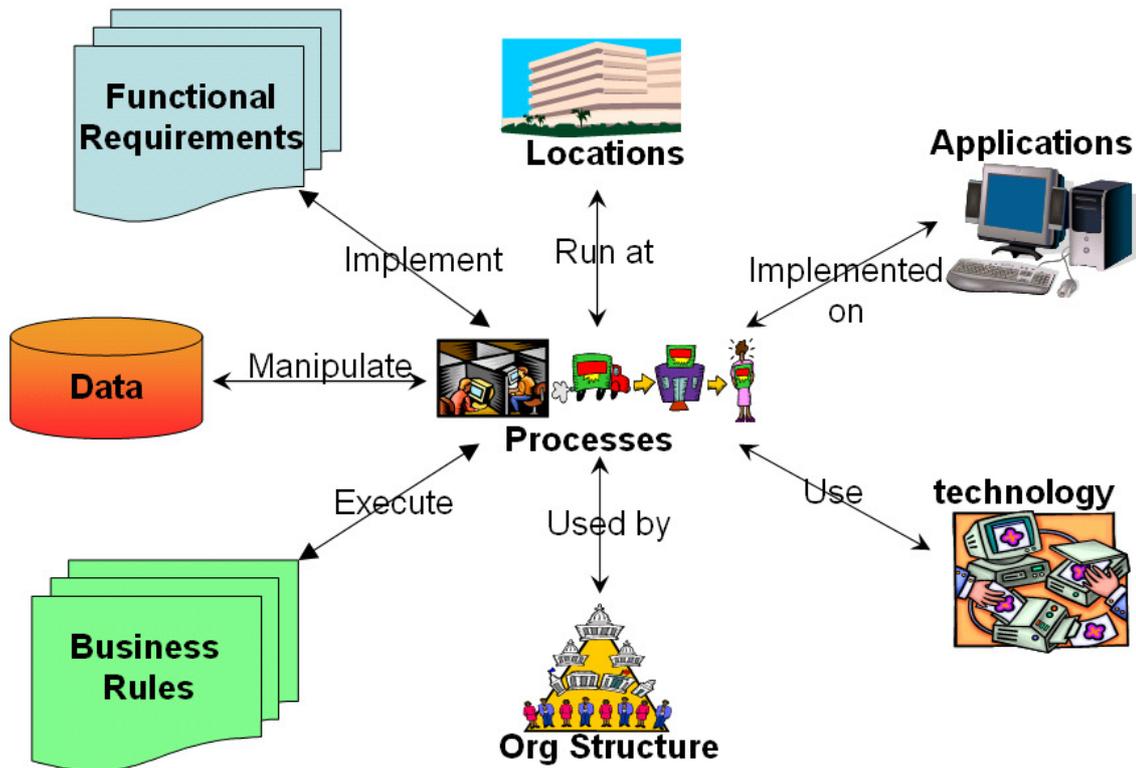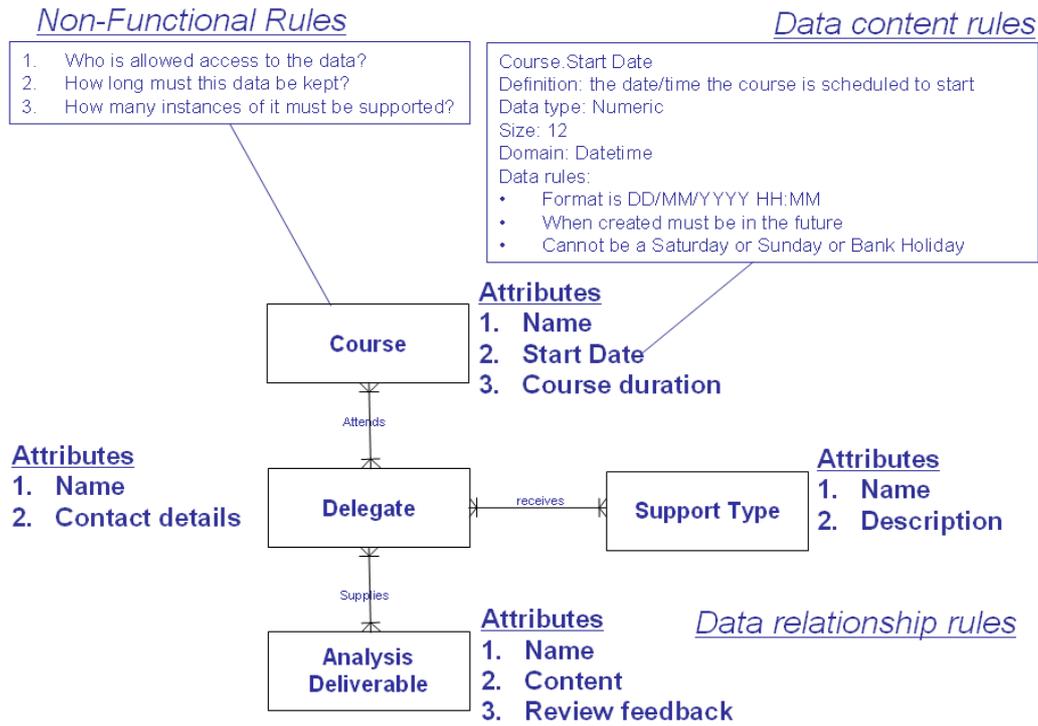

Data models are the pivot around which processes are specified to create, read, update and delete data. This means that data specifications, while not the most visible of analysis deliverables to users, are central to a successful solution and poorly specified data requirements inevitably result in cumbersome processes to manipulate them.

It is worth realising that all solutions (computerised or not) will have requirements connected with the information they need (data requirements) whether they know it or not!

Data models (the drawings of data and relationships between them) are only one facet of the specification of a data. These are the other facets:

Non-Functional Rules

1. Who is allowed access to the data?
2. How long must this data be kept?
3. How many instances of it must be supported?

Data content rules

Course.Start Date
Definition: the date/time the course is scheduled to start
Data type: Numeric
Size: 12
Domain: Datetime
Data rules:
• Format is DD/MM/YYYY HH:MM
• When created must be in the future
• Cannot be a Saturday or Sunday or Bank Holiday

Course

Attributes
1. **Name**
2. **Start Date**
3. **Course duration**

Attends

Attributes
1. **Name**
2. **Contact details**

Delegate

receives

Support Type

Attributes
1. **Name**
2. **Description**

Supplies

Analysis Deliverable

Attributes
1. **Name**
2. **Content**
3. **Review feedback**

Data relationship rules

This article will look at the reasons why data modelling should be done (the benefits) and the techniques for
• drawing data models
• documenting data content rules
• documenting data non-functional requirements

Finally – some of the most useful advanced data modelling tools will be outlined.

Data models - concepts.
We will use Entity Relationship Diagrams (ERD) as this is the most widely used. Class diagrams are also becoming popular.

ERDs have an advantage in that they are capable of being normalized. There is a whole specialism normalization, however this article will only work on models to 3rd normal form which is sufficient for most purposes.

3rd normal form can be summarized as "the minimum amount of data recorded the minimum number of times"

3rd normal form reduces redundancy and room for error in specification of requirements and development of a solution.

A data model consists of entities related to each other on a diagram:
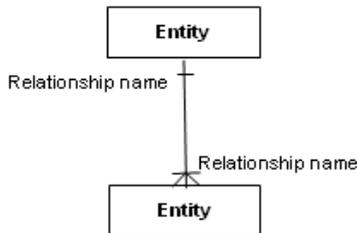
| Data model element	Definition
Entity	A real world thing or an interaction between 2 or more real world things.
Attribute	The atomic pieces of information that we need to know about entities.
Relationship	How entities depend on each other in terms of why the entities depend on each other (the relationship) and what that relationship is (the cardinality of the relationship).

Example:

Given that …

- "Customer" is an entity.

- "Product" is an entity.

- For a "Customer" we need to know their "customer number" attribute and "name" attribute.

- For a "Product" we need to know the "product name" attribute and "price" attribute.

- "Sale" is an entity that is used to record the interaction of "Customer" and "Product".

Here is the diagram that encapsulates these rules:

smart·BA
distance learning programme
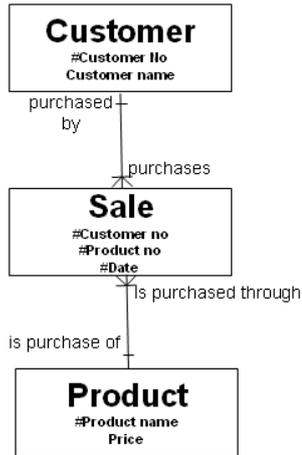
Notes:
1. By convention, entities are named in the singular.
2. the attributes of "Customer" are "Customer No" (which is the unique identifier or primary key of the "Customer" entity and is shown by the # symbol) and "Customer Name".
3. "Sale" has a composite primary key made up of the primary key of "Customer", the primary key of "Product" and the date of the sale.
4. Think of entities as tables, think of attributes as columns on the table and think of instances as rows on that table:

Customer *(entity)*

No *(attribute)*	Name *(attribute)*	
10	Fred Bloggs	*(instance)*
67	Freda Jones	*(instance)*

Sale

| Customer No	Product Code	Date
10	101	21/2/2020
67	452	22/2/2020

Product

Code	Name	Price
101	Flange	£123.00
452	Blitwort	£34.50

5. If we want to know the price of a Sale, we can 'find' it by using the "Product Code" on the instance of "Sale" we are interested in and look up the corresponding "Price" on the "Product" entity with the matching "Product Code".

Relationship cardinality.
This term refers to what the relationship dependency is and the valid set of cardinalities is shown here:
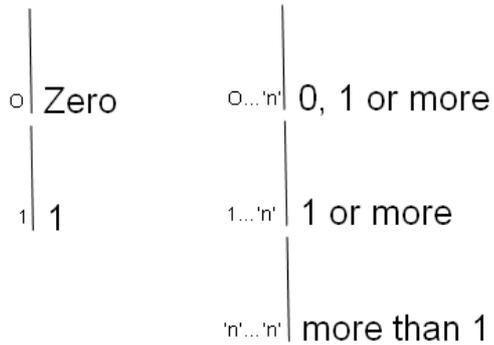
Crows Foot Cardinality

Zero or 1

0, 1 or more

1

1 or more

more than 1
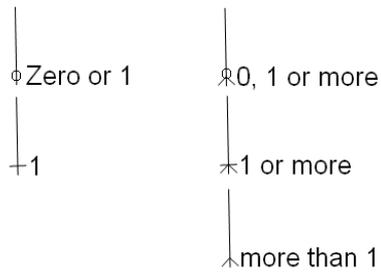
This is known as crows foot notation.

There are other notations such as UML Class Associations:

UML Class Cardinality

o Zero

1 1

o...'n' 0, 1 or more

1...'n' 1 or more

'n'...'n' more than 1

Whatever notation, cardinality is defined at both ends of a relationship and in any combination.

Cardinality refers to the entity that it is closest to.

smart·BA
distance learning programme

Reading a data model.

Relationships are read by building a sentence:

a.   Name the entity where you are starting from in the singular: "Each Customer"
     (Note: entity names by convention start with an upper case letter).

b.   Read the name of the relationship you are reading and put it in to a sentence: "purchases"
     (Note: the end of the relationship the name is at is arbitrary: construct the best sentence you can in
     context of the requirements as you understand them).

c.   Read the cardinality of the end of the relationship you are reading: "1 or more"

d.   Name the entity you are ending up on in the singular or plural depending on the cardinality at that end
     of the relationship: "Sales".

All together this makes: "One Customer purchases one or more Sales".

In this case as the Customer must purchase at least one sale you can (and should) include this in the
sentence: "One Customer **must** purchase one or more Sales".

If the Customer did not have to purchase any Sales to be a Customer then the sentence would be "One
Customer **may** purchase zero, one or more Sales".

When you are reading a data model like this you have to be able to contextualise it in terms of requirements
that are being satisfied by the data model.

So, in the example we have of "One Customer must purchase one or more Sales" we would want to be sure
that this *is* the business rule.

Example reading of a data model:
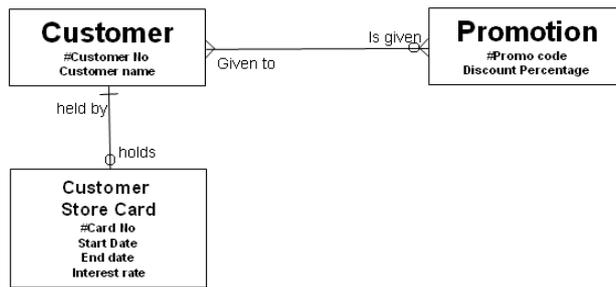
1. Each Customer may be given zero, 1 or more Promotions. (Note: the use of the word 'may' here indicates that this is an optional relationship in that the customer may have zero Promotions or one or many Promotions).
2. Each Promotion must be given to more than 1 Customers. (Note: the use of the word 'must' here indicates that the relationship is mandatory: if a Promotion exists then it must be given to more than 1 Customer).
3. Each Customer may hold zero or 1 Customer Store Cards.
4. Each Customer Store Card must be held by 1 Customer.

Notes:
1. **Always** use the full name of the entity at each end of the relationship you are reading.
2. **Always** use 'may' for relationships that have the option of zero for the target entity.
3. **Always** use 'must' for relationships that do not have the option of zero for the target entity.
4. **Always** use the relationship name (if there is one) or create one from what you know of the requirements.
5. **Always** make sure that the relationship name represents a business requirement.
6. The sentences should be grammatical.
7. Sense check: would you predict a business representative would be able to validate the sentence you construct from the data model as being a business requirement?
8. **Never** use database design terminology (this is a definition of requirements for data, not a design for a database!).

Building a data model - overview.

There is a process to go through which will be described in more detail in the next section:
1. identify candidate entities
2. select a central candidate entity
3. define the primary key
4. work through the rest of the candidate entities:
    a. consider whether it is in scope
    b. define the primary key
    c. define the relationship(s) between the candidate entity and all other candidate entities on the diagram
    d. add description entities as needed
    e. fully express the cardinality of the relationship(s)
    f. name the relationship(s)
5. for each entity on the diagram
    a. define the remaining attributes
    b. define the non-functional requirements
6. review and refine
7. Advanced data modelling concepts
    a. Super-types and sub-types
    b. Hierarchies
    c. Exclusive relationships
    d. 3rd normal form

Building a data model - detail.

There is a process to go through:
  1. identify candidate entities

   Remember: an entity is a real world thing or an interaction between real world things.

   There are a number of techniques that can be used to identify candidate entities such as interviewing, reading background documentation, reading process model specifications, and so on. They all require that nouns (not proper nouns which are the names of individual instances of nouns) are indentified. Examples of nouns: customer, product, sale.

  2. select a central candidate entity
   It does not really matter which entity is chosen as all entities on a diagram will link to at least one other.

   Typically, an entity relationship diagram for a normal sized project will have 20-30 entities. There will normally be just a few of these entities that a lot of the others relate to. Try to pick one of these and good clue will often be in the project name: e.g. Customer Sales Oder System would suggest Customer and Sales Order. However, it does not make any material difference to the outcome of the data modelling exercise which entity is selected.

3. **define the primary key**

   Remember: a primary key is an identifier than uniquely identifies one instance of an entity.

   Primary keys – once assigned to an instance of an entity – cannot be modified, deleted or changed in any way.

   Because of this, the primary key of an entity is – in fact – what the entity actually *is* as opposed to what it is called. Suppose there is an entity called Person and the primary key is Employee No.

   What this means in fact is that the entity should be called Employee as this entity is incapable of holding information about anyone who does not have an employee number and the only people who have employee numbers are employees.

   A primary key should be a piece of business information – such as employee number. That is what people use to identify employees and that is what uniquely identifies one employee from another.

   Avoid primary keys that mean nothing to the business and they do not use (example: a "system assigned key"). If an entity is – in fact – what its primary key uniquely identifies then having a "system assigned key" means either the entity has no business identifier (so it quite literally is nothing) or that what is being defined is a database table and not a business entity.

4. **work through the rest of the candidate entities:**
   Work your way through the rest of the list of candidate entities. For each one…
   a. **Consider whether it is in scope**
      Just because an entity does exist does not mean it is in scope of the work you are analysing data requirements for. There is a simple test to apply to decide this question: does at least one process in scope use it (already documented or not)? If not, either a process is missing or it is out of scope!

   b. **define the primary key**
   Refer to point 3 above.

   c. **define the relationship(s) between the candidate entity and all other candidate entities on the diagram**
      You will find that the entity you are adding to the diagram has a direct or indirect relationship with at least one entity already on the diagram. If the relationship is indirect it will be via other entities that you may have to bring off your list of candidate entities. If it does not link then either you are missing some entities that would create an indirect link or it is out of scope of your current work. Remember, if a process does use it **is** in scope and that means you **are** missing some other entities!

smart·BA
distance learning programme

d. Add description entities as needed
There is a class of entities known as description entities. These entities classify other entities. Their job is to control what classifications are allowed. Example:
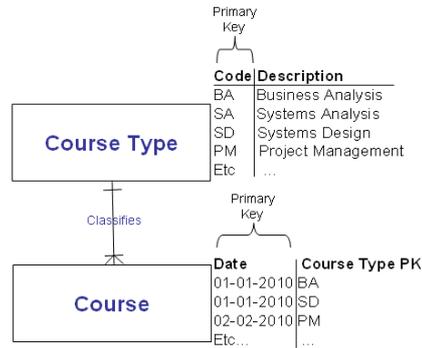
The primary key of Course Type is a code called PK.

The only other attribute is a description of the code.

Frequently this is all that the description entities contain, though they can and do contain other attributes when required. In this example we could – if it was in scope – store the standard price of the Course Type as an attribute.

The primary key of Course is a composite key:
- the date of the course
- the type of the course. Note that this is the PK of the descriptor entity.

The attributes of the Course Type must be business data – so that means the code that the business knows the course by. If they know it by the full name then that should be used.

e. fully express the cardinality of the relationship(s)
Resolve many to many.
Consider the following
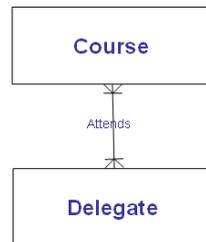
A Delegate can *attend* 1 or more Courses.
A Course will be *attended by* 1 or more Delegates.
Which Delegates are attending which Course?

Answer 1: It would be possible to answer this by holding the courses attended by the Delegate on the Delegate entity:

Delegate Pk	Course1	Course2	Course3

Problems:
- If a Delegate attends more than 3 Courses another attribute would be needed.
- How many are needed? If the business rule is Delegate can attend as many Courses as they like, then potentially an infinite number are needed?
- If more information about the Courses a Delegate attends is needed (e.g. date of enrolment, special needs for the course, etc) then more attributes *per Course attribute* will be needed.
- How does the question "how many delegates are there on the BA course for 01-01-2010?" get answered? It would be necessary to look in all the course attributes to count up instances of that Course.
- It is not 3rd Normal Form as now not every attribute will depend on the primary key only.

**Answer 2: Alternatively store the Delegates on the Course entity.**

Course Pk	Delegate1	Delegate2	Delegate3

The problems with this are the same as with storing the Courses on the Delegate entity.

Answer 3: Do both Answer 1 and Answer 2!
This produces the same problems and adds the fact that it would be possible to record a Delegate007 attending Course1 on the Delegate entity, and Course1 **not** having Delegate007 on the Course entity!

**The correct answer is to create an intersection entity:**
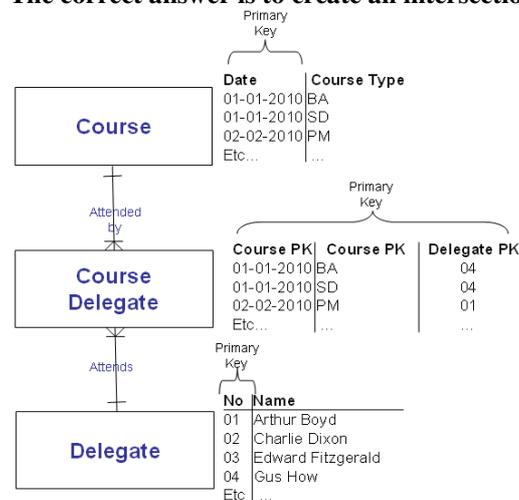
It is normally possible to name the intersection entity by concatenating the names of the 2 entities in the many:many relationship in to a meaningful (for the business) name.

There are exceptions to this. For example, the intersection entity resulting from the many:many between Product and Customer is Sale.

Essentially, these intersection entities record interaction events between the 2 entities in the many:many relationship and hold at the very least the valid combinations of the 2 entities.

Combine one to one.
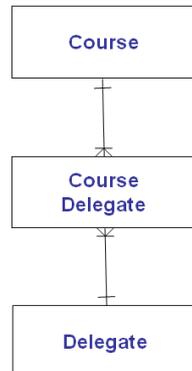Far less common is the scenario where there are 2 entities in a one:one relationship.
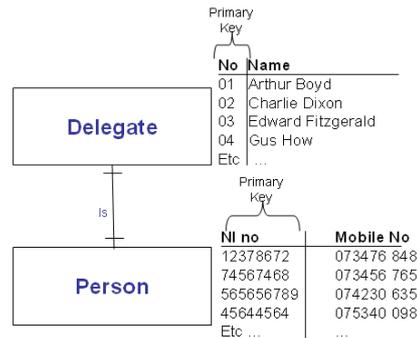
This shows that a Delegate is a Person and a Person is a Delegate. The name of the Delegate changes if we look at a different NI No. What this is saying is

- Delegate Name depends on the Person NI No.
- NI No could be used as PK for Delegate and Delegate Init could be used as a PK for Person.

In this situation the 2 entities could be collapsed in to 1 entity with a PK of Init or NI No.

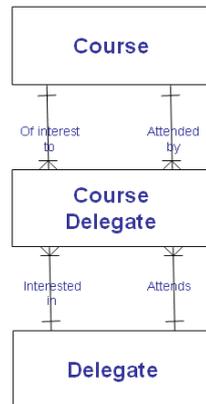**f.   name the relationship(s)**
Consider the following:

We could assume that this shows the courses that delegates *attend.* But it is equally valid to state this shows which courses delegates are *interested in.*

The *italics* are the names of the relationships.

The section on **reading a data model** covers how the relationships should be named.

Bear in mind that in the above case it may be true that both relationships are in scope. If so, then show them:

Note that while in theory it is correct to name the relationships in both directions so that sentences can be built up starting with any entity, in practice it is only necessary to name the relationship in one direction as – with a little practice – naming the relationship in the other direction becomes intuitive once the name is known in one direction.

5. **for each entity on the diagram**
    a. **define the remaining attributes**
        The Primary Key of an entity is either one or more attributes of that entity, one or more relationships that entity takes part in (so has the primary keys of some other entities which are themselves attributes) or a combination of attributes and relationships.

        However, there will be other information processes need to know about (for example) Delegates other than the Primary Key – the Delegate's Date Of Birth for example. These other items are the other attributes. If you are not sure if an item is an attribute or a entity, try asking yourself whose item of information it is.

        Example: 'Name' – is this an attribute or an entity? Well, the Name of what? If the Delegate's Name then it is an attribute of Delegate, if it is the Course Name then it is an attribute of Course.

        Another example: 'Company' – is this an attribute or an entity? Well, we cannot ask "the Company of what?" they just are Companies. This is an entity and you would want to know how Companies are uniquely identified: what their primary key is.

Once all the attributes have been identified, it is necessary to specify them (including the Primary Key except for Primary Key elements that are relationships as these attributes will be specified on the entities where they are Primary Keys):

- Definition: what, unambiguously, is this attribute? Define it in Business Terms.
- What type of data must it be capable of holding? Free text, numbers, dates, etc.
- How big does the attribute need to be. Remember you are specifying requirements, not computer database fields. Thus a Date attribute will *need* to be capable of holding 8 numbers (YYYYMMDD). A Comment attribute may have no size limit that the users are aware of so that should be specified: size – infinite. The system designers will tell you if this is possible or not, but if you don't even ask if it is possible you will never know.
- Domain. A domain is not applicable to most attributes. It is set of valid values (sometimes quite a large set) that the attribute can hold. This set of values has certain characteristics:
  - ➢ There is only one type of item in the value set. Example: date is a domain. The only type of value in the date domain is dates. If we also wanted the week no of the date, this would not work.
  - ➢ These is a list of valid items for the domain. Conceptually there is a list of valid dates (28/02/2004 is valid, 29/02/2004 is not) – this may be a large list as it is in dates, but it is still a list.
  - ➢ Domain values themselves are never added to, removed or changed. No new dates will ever be created, they will never be removed and they will never be changed (28/02/2004 will always be just that).
- Data rules: for a Date attribute you might want to specify that it must be held in the format YYYYMMDD or DDMMYYYY. For a Comment attribute you might need to specify that swear words are not permitted – and list the words to be excluded! Any rule specified must always be applicable to that attribute. Any rules that depend on the stage in the process should be specified in the process specifications.

smart·BA
distance learning programme

b. **define the non-functional requirements**

What are non-functional requirements? They are requirements that are not functional! This means they include everything that cannot be defined as functional (as providing a capability or function in some way). In that sense they are a 'catch-all'.

For data models the non-functional requirements apply at entity level and could include

- Who is allowed to access the entity?
- How long must the data be kept?
- How many instances must it be possible to keep?

Example: Customer entity possible non-functional requirements:

- only sales people who work at the shop where the Customer has made a purchase can access the Customer data.
- Customer data must be kept 6 years.
- At any one point in time there may be 250,000 Customers max.
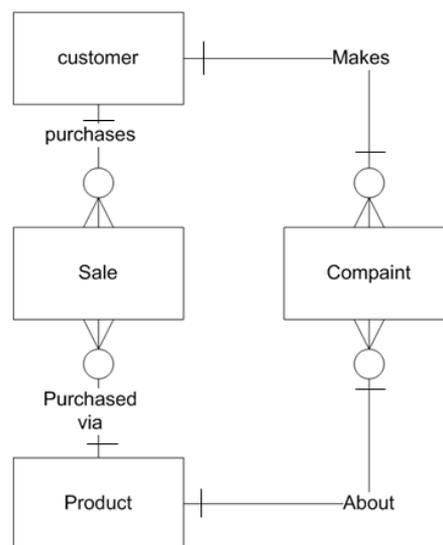
6. **review and refine**
   Quality checks:
   a. **Primary Keys**
      - Does every entity have one?
      - Have they been described/defined?
      - Do the primary keys conform to the rules for primary keys (i.e. uniquely identify instances of the entity, never change, and the way the way the *business* identify the entity)?
   b. **Circular relationships between entities**
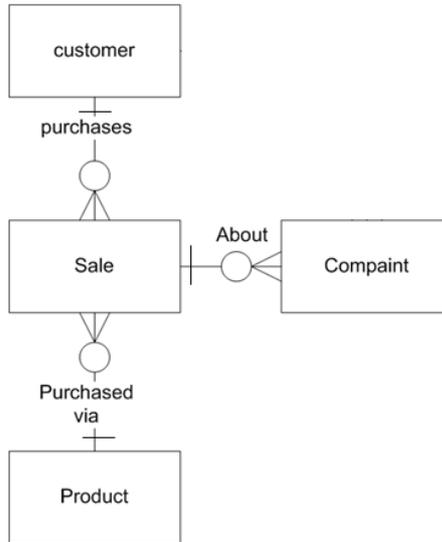   - **Consider the following diagram:**

      -
      - There is a circular relationship involving all 4 entities. This means the diagram supports the following navigation: from Sale to the Customer who purchased

it, from Customer to all Complaints they have made, from Complaint to the Product it is about and from Product to all Sales of that Product.

- The problem is that this allows a compliant to be made by a Customer for a Product that is NOT one of the Products on the Sales entity for that Customer. Is it right that a customer can complain about a Product they have not purchased? Probably not.

- Here is how to fix it:

-
- Now a Complaint MUST be in connection with a Sale for a Product the Customer has purchased!
- Howver, consider the following circular relationship:

-
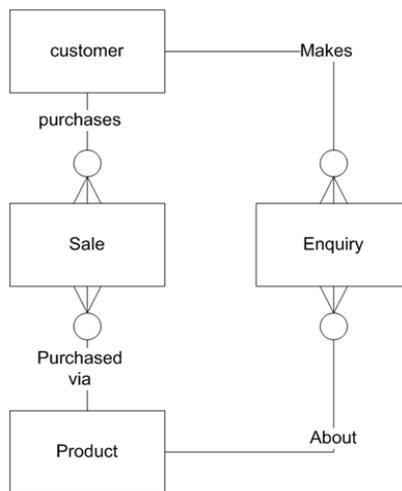
- How should this be fixed? Arguably it should not: the business may WANT a Customer to be able to Enquire about a Product they have not already purchased via a Sale.
- You have to assess if a circular relationship creates a logical inconsistency (a Customer can Compalin about a Product they have not purchased via Sale) or meets requirements (a Customer can Enquire about a Product they have not yet purchased via a Sale).

c. Many to Many
- Have all many:many relationships should be resolved via intersection entities?

d. One to One
- Have all one:one relationships been consolidated to one entity?

e. Attributes specification
- Have all attributes been identified?
- Have they been described/defined?
- Have they been specified (data type, size, domain, rules)?

7. Advanced data modelling concepts
There are many. Here are some of the most useful.

a. Super-types and sub-types
- Entities can be categorised using Type entities.
  Example: Customer has a Customer Type entity of "Employee" and "Non_Employee" to support the requirement that employees of a bank can also be customers of that bank.
  However, consider the following business rules that also apply:
  1. Customers must have a No and Name (No is the primary key)
  2. when the Customer is of type "Employee" then record their employee number, who they report to, and job title.
  3. Customer of type "Non_Employee" must only have a contact telephone number.

  Given that
  1. There are at least 2 types of the entity Customer
  2. They have the same primary key
  3. Which type the Customer is determines what attributes the Customer entity needs and
  4. what relationships it participates in

 Then the following supertype/subtype construct will work:
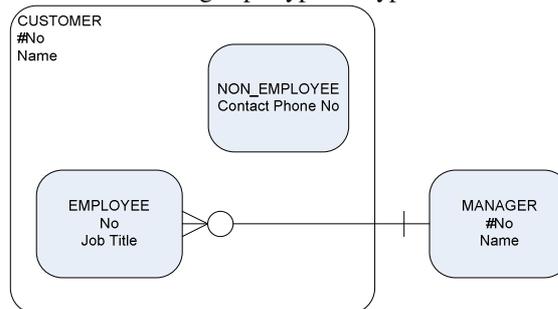
Employee has the attributes of Customer (critically the primary key) as well as the attributes Employee No and Employee Job Title.

Non_Employee also has the attributes of Customer (critically the primary key) as well as the attribute Contact Telephone No

b. Hierarchies

- The following information needs to be recorded: Team, the Department the team is part of, the Division the Department is part of. This could be modelled with 3 entities (Team, Department and Division). Suppose that we only need to know the name and description of the Team, Department and Division. In this case, a recursive relationship (or pig's ear as it is often called) can be used:

Part of

ORGANISATION
UNIT
#Name
Description
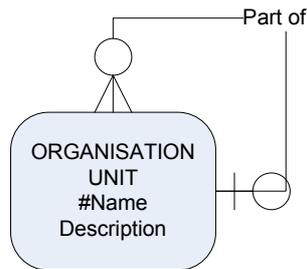
-
- It is very common for entities to be in hierarchies.
- Note that these are almost always optional at both ends of the relationship, otherwise it would be impossible to get to the top or bottom of the hierarchy!

c. Exclusive relationships

- Suppose a business requirement exists that a customer's compliant can be about products they have purchased or services they have received, but never both for one complaint.
- If it is true that Products and Services have different primary keys then the following construct can be used (the bar across the 2 relationships to Product and Service from Complaint):

CUISTOMER
#No

Makes

COMPLAINT
#Customer PK
#ProductORServicePK
#Date

About

About

PRODUCT
#No
Name
TechSpec

SERVICE
#Name
Description

-

smart·BA
distance learning programme