**Design databases**

1. **Identifying Entities**

* Dans le développement d’un projet, la première chose à faire est de savoir les acteurs qui vont participer dans ce projet, et plus précisément, les entités qui vont être actif.

**Exemple :**

On va définir les acteurs pour un projet de site web d’un magasin ; on doit recueillir les informations avec laquelle on va travailler.

* Dans un magasin, on vend des produits aux clients
* Le magasin est une localisation
* Vendre est un événement
* Produit sont des objets
* Les clients sont des humains

Toutes ces entités formeront le besoin qui va être inclus dans la base de données.

Mais quels sont les autres informations qui se produisent lors de la vente d’un produit ? Tout ceci doit être englobé afin de compléter notre conception.



1. **Identifying Relationships**

* La prochaine étape maintenant est de déterminer la relation entre les différentes entités acteur de ce système. la relation est qu’est ce qu’une entité fait avec une autre.

🡺 Donc faut écrire les scénarios qui se passent entre les différentes entités entre eux.  
🡺 Les cardinalités sont combien dans un coté de la relation apporte a l’autre coté de la relation.

**Exemple :**

Combien de clients dépendent d’une seule vente ? Combien de ventes dépendent d’un seul client ? Combien de ventes dépendent d’un seul magasin ?

On aura comme ceci :

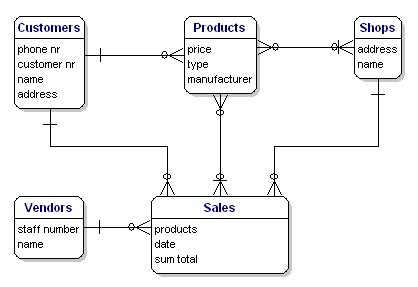
* **Customers --> Sales;** 1 customer can buy something several times
* **Sales --> Customers**; 1 sale is always made by 1 customer at the time
* **Customers --> Products;** 1 customer can buy multiple products
* **Products --> Customers;** 1 product can be purchased by multiple customers
* **Customers --> Shops**; 1 customer can purchase in multiple shops
* **Shops --> Customers**, 1 shop can receive multiple customers
* **Shops --> Products**; in 1 shop there are multiple products
* **Products --> Shops;** 1 product (type) can be sold in multiple shops
* **Shops --> Sales**; in 1 shop multiple sales can me made
* **Sales --> Shops;** 1 sale can only be made in 1 shop at the time
* **Products --> Sales**; 1 product (type) can be purchased in multiple sales
* **Sales --> Products;** 1 sale can exist out of multiple products

**Est ce qu’on a mentionnée toutes les relations ?** Au faite Non ! Car une relation réfère à une règle de gestion, donc tout dépend de ce que l’on souhaite réaliser, on pourra l’insérer dans notre schéma.

**Comment faciliter la schématisation ?** On va mettre les cardinalités par chaque relation  
🡺**Customers --> Sales;** 1 customer can buy something several times  
🡺**Sales --> Customers;** 1 sale is always made by 1 customer at the time

So we’ll get:

* **Customers --> Sales**; --> 1:N
* **Customers --> Products**; --> M:N
* **Customers --> Shops**; --> M:N
* **Sales --> Products**; --> M:N
* **Shops --> Sales**; --> 1:N
* **Shops --> Products**; --> M:N



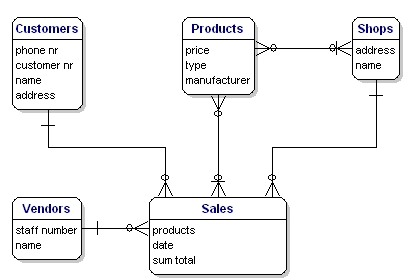
1. **Simplify the diagram**

### Redundant Relationship

Sometimes in our model, we can get a ‘redundant relationship’, There are relationships that are already indicated by other relationship, although not directly.

In the case of our example, we have a direct relationship between customers and products. But there are also relationships from customers to sales and from sales to products, so indirectly we have a relationship between customers and products through sales.

The new model will then look like:

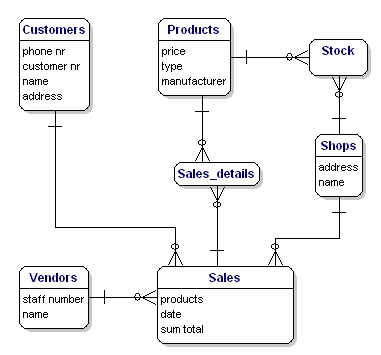


**Solving Many-to-Many relationship**

Many-to-many relationships (M: N) are not directly possible in a database.  Somewhere you need to save which records these are and the solution is to split the relationship up in two one-to-many relationships.

**Products <----> Sales**: Every sale includes more products. The relationship shows the content of the sale. In other words, it gives details about the sale. So the entity is called 'Sales details'. You could also name it 'sold products'.

**Products <----> Shops**: Shows which products are available in which the shops, also known as 'stock'. Our model would now look like this:



1. **Identifying Attributes**

The data elements that you want to save for each entity are called 'attributes'.  
Attributes are all the information that we want to know about an entity.

For example:

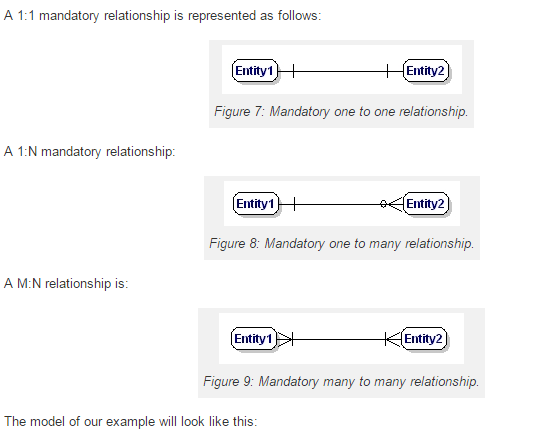
* **About the products that you sell**: you want to know, for example, what the price is, what the name of the manufacturer is, and what the type number is.
* **About the sales**: you know when they happened, in which shop, what products were sold, and the sum total of the sale.

**Derived data:**

Derived data is the data that is derived from another data that we have already saved. In this case of the ‘sum total’ is a classical case of derived data. We know exactly what has been sold and what each product costs. So we can calculate how much the ‘sum total’ is, so it’s not necessary to save the sum total.

So why is it saved here? Well, because it is a sale, and the price of the product can vary over time.   
A product can be priced at 10 Euros today and at 8 Euros next month and for your administration you need to know what it cost at the time of the sale, and the easiest way to do this is to save it here. There are a lot of more elegant ways, but they are too profound for this article.

1. **Presenting Entities and Relationships: Entity Relationship Diagram (ERD)**

****

