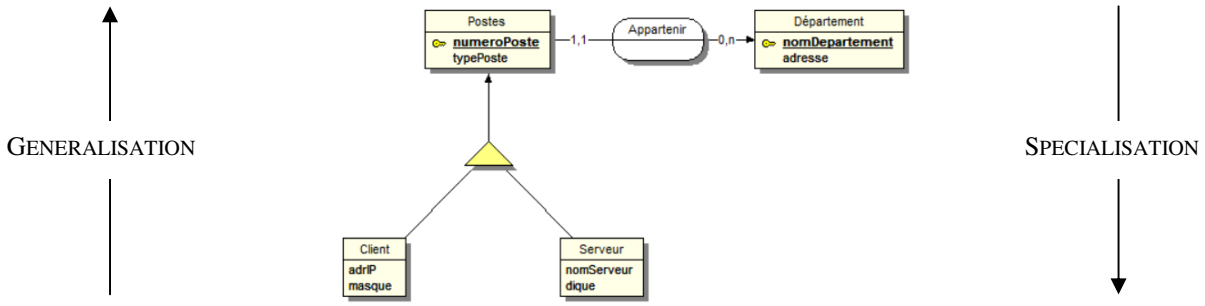
# L'héritage dans le Modèle Entité/Association

## 1 L'héritage

Tout comme en programmation orientée objet, il est possible d'utiliser l'héritage dans un modèle entité-association. L'héritage permet de faire de la spécialisation ou de la généralisation. La spécialisation consiste à dériver une entité déjà existante pour créer une nouvelle entité qui en plus des caractéristiques (propriétés et associations) de la super-entité possède des caractéristiques supplémentaires. Alors que la généralisation consiste à mettre en facteur dans une super-entité les caractéristiques qui sont communes à un ensemble d'entités.




Il est à noter que les sous-entités héritent des propriétés mais aussi des associations de la super-entité. Par exemple, dans l'exemple ci-dessus, un serveur hérite de toutes les caractéristiques d'un poste. C'est-à-dire, ses propriétés (numéro de poste, type de poste) mais aussi du département auquel il appartient. Cela veut dire que si on rajoute une association entre Serveur et Département (pour indiquer à quel département appartient le serveur), le Serveur aura alors deux départements !

## 2 Décomposition de l'héritage dans le Schéma Relationnel

Il existe trois façons de décomposer un héritage dans un Schéma Relationnel : la décomposition ascendante, la décomposition descendante et la décomposition par distinction. Chacune de ces décompositions a ses avantages et ses inconvénients. Le concepteur doit choisir une de ces décompositions en fonction du contexte général de l'étude. Il doit notamment prendre en compte le nombre d'associations qui sont rattachées aux entités de la hiérarchie (y a-t-il beaucoup/peu d'associations sur la super-entité ; y a-t-il beaucoup/peu d'associations sur les sous-entités), les traitements (requêtes) qu'il faudra réaliser par la suite dans la base de données, ou encore, comme nous le verrons plus tard, la contrainte qu'il y a sur l'héritage (eXclusion, Totalité, Partition).

### 2.1 La décomposition ascendante

Avec cette décomposition on stocke toutes les données dans une seule table qui correspond à la super-entité de la hiérarchie. Cette solution a le mérite de simplifier le Schéma Relationnel mais d'un autre coté elle multiplie les `NULL`.

POSTE (numeroPoste, type, adrIP, masque, nomServeur, disque)

```
CREATE TABLE Poste    (numeroPoste VARCHAR(4), type VARCHAR(15), adrIP VARCHAR(12), masque VARCHAR(12),
                                                                    nomServeur VARCHAR(20), disque VARCHAR(1),
                       CONSTRAINT pk_Poste PRIMARY KEY (numeroPoste));
```

### 2.2 La décomposition descendante :

Avec cette décomposition on crée autant de tables qu'il y a de sous-entités. Cette solution est adaptée lorsque on ne peut pas avoir d'occurrence de la super-entité (c'est à dire qu'elle est comme abstraite) et que l'on ne peut pas avoir non plus d'occurrences qui proviennent de plusieurs sous-entités à la fois. Dans le cas contraire, cette décomposition est à éviter.

CLIENT (numeroClient , type, adrIP, masque)
SERVEUR (numeroServeur, type, nomServeur, disque)

CREATE TABLE Client (numeroClient VARCHAR(4), type VARCHAR(15), adrIP VARCHAR(12), masque VARCHAR(12),
CONSTRAINT pk_Client PRIMARY KEY (numeroClient));
CREATE TABLE Serveur (numeroServeur VARCHAR(4), type VARCHAR(15), nomServeur VARCHAR(20), disque VARCHAR(1),
CONSTRAINT pk_Serveur PRIMARY KEY (numeroServeur));

### 2.3 La décomposition par distinction :

Avec la décomposition par distinction, on crée une table pour chaque entité de la hiérarchie. Cette décomposition est la décomposition la plus flexible. Elle a l'avantage de pouvoir être utilisée dans tous les cas de figure ce qui fait d'elle la décomposition la plus utilisée en pratique. Par contre elle nécessite d'utiliser un plus grand nombre de tables.

POSTE (numeroPoste, type)
CLIENT (numeroClient#, adrIP, masque)
SERVEUR (numeroServeur#, nomServeur, disque)

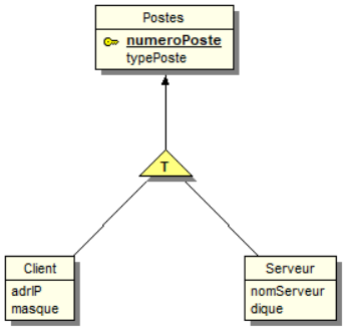
CREATE TABLE Poste (numeroPoste VARCHAR(4), type VARCHAR(15),
CONSTRAINT pk_Poste PRIMARY KEY (numeroPoste));
CREATE TABLE Client (numeroClient VARCHAR(4), adrIP VARCHAR(12), masque VARCHAR(12),
CONSTRAINT pk_Client PRIMARY KEY (numeroClient),
**CONSTRAINT fk_Client_numeroClient FOREIGN KEY (numeroClient) REFERENCES Poste(numeroPoste)**);
CREATE TABLE Serveur (numeroServeur VARCHAR(4), nomServeur VARCHAR(20), disque VARCHAR(1),
CONSTRAINT pk_Serveur PRIMARY KEY (numeroServeur),
**CONSTRAINT fk_Serveur_numeroServeur FOREIGN KEY (numeroServeur) REFERENCES Poste(numeroPoste)**);

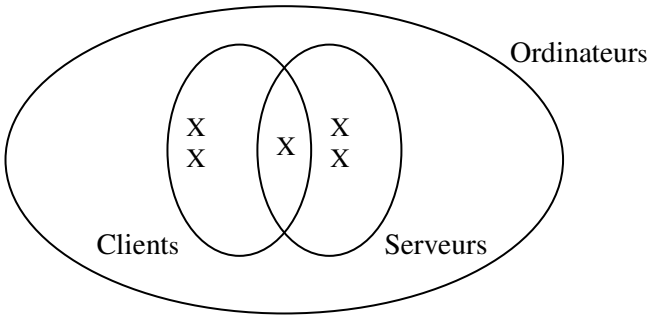
## 3 Contraintes sur les héritages

Il est possible de représenter une contrainte sur un héritage afin d'indiquer s'il est possible ou non d'avoir une occurrence de la super-entité, ou bien s'il est possible ou non qu'une occurrence corresponde à plusieurs sous-entités à la fois.

### 3.1 La contrainte de Totalité

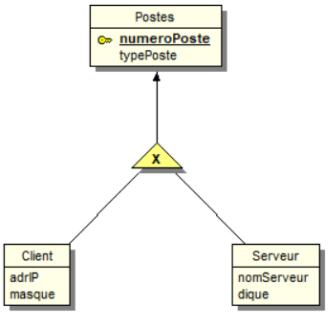
La contrainte de Totalité permet d'empêcher qu'une super-entité puisse avoir des occurrences propres. On la représente avec un T dans le modèle entité/association. En programmation objet, cela correspond à dire que la super-classe est abstraite.
Dans notre exemple, si on indique une contrainte de Totalité, cela veut dire qu'on ne peut pas avoir de poste isolé qui n'est ni client ni serveur. Par contre, il est possible d'avoir des clients, des serveurs, ou bien des client-serveur.

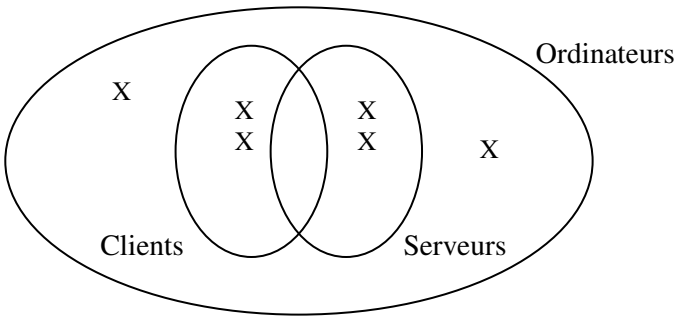



### 3.2 La contrainte d'eXclusion

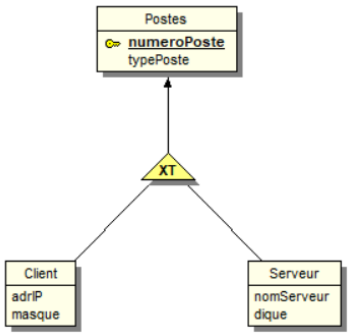
La contrainte d'eXclusion permet d'empêcher qu'une occurrence corresponde à plusieurs sous-entités à la fois. On la représente avec un X dans le modèle entité-association. En analyse orientée objet cela signifie que la classification multiple (un objet est l'instance de plusieurs classes à la fois) n'est pas possible. En programmation Objet, comme il n'est pas possible d'implémenter la classification multiple, cette contrainte est inutile. Par contre on pourra la représenter sur un modèle entité-association quand cela est nécessaire car dans une base de données relationnelle il est possible d'implémenter la classification multiple.
Dans notre exemple, la contrainte d'eXclusion indique qu'on ne peut pas avoir de client-serveur. Par contre il sera possible d'avoir des clients, des serveurs ou des postes qui ne sont ni client ni serveur.

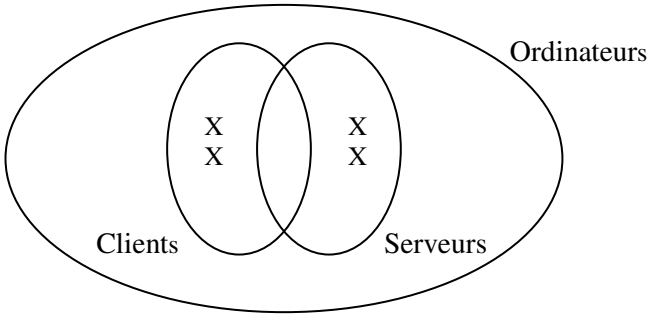



## 3.3 La contrainte de Partition

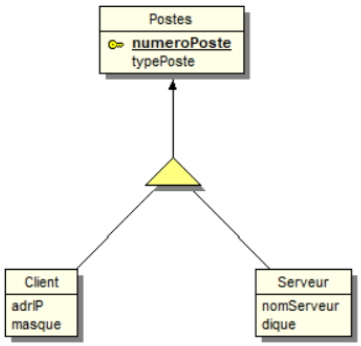
La contrainte de Partition permet d'indiquer qu'on a à la fois une contrainte d'eXclusion et une contrainte de Totalité. On la représente donc avec un XT dans le modèle entité-association.
Dans notre exemple, la contrainte de partition (XT) indique qu'on peut avoir uniquement des clients ou des serveurs. Il n'est pas possible d'avoir des client-serveur (X) ni des postes isolés (T).

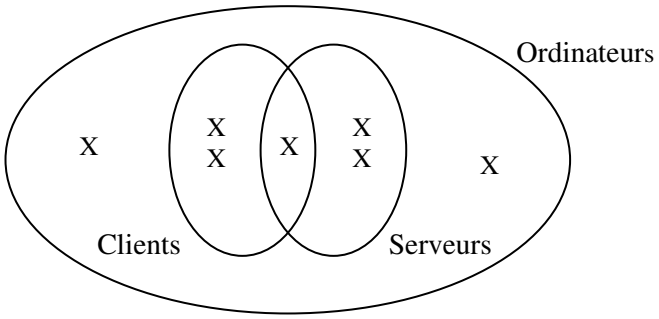



## 3.4 Héritage sans contrainte

Il est à noter que si un héritage ne possède pas de contrainte il est alors possible d'avoir toutes les cas de figure.
Dans notre exemple, si on n'a pas de contrainte, on peut avoir des clients, des serveurs, des postes ni client ni serveur, et des client-serveur.




## 3.5 Importance des contraintes sur le choix de la décomposition

Comme indiqué précédemment, le choix de la décomposition dans le schéma relationnel va dépendre de plusieurs critères dont le nombre d'associations qui sont rattachées à la super-entité et aux sous-entités. Ce choix peut aussi dépendre de la contrainte qui est indiquée sur l'héritage. Par exemple, avec une contrainte de Totalité (T), il peut être intéressant d'utiliser une décomposition descendante (sous réserve qu'il n'y ait pas trop d'associations qui sont reliées à la super-entité).