

# Conception de bases de données

Pierre Lefebvre

# A quoi sert une méthode ?

**Une méthode définit une démarche reproducible qui produit des résultats fiables.**

**Une méthode d'élaboration de logiciels décrit comment modéliser et construire des systèmes logiciels de manière fiable et reproducible.**

**De manière générale, une méthode définit :**

- ☞ Des éléments de modélisation,
- ☞ Une représentation graphique,
- ☞ Du savoir-faire et des règles

**Avec, en autre, les objectifs suivants :**

- ☞ Se donner toutes les chances de mener à bien un projet informatique,
- ☞ Établir un plan projet réaliste en définissant, estimant et planifiant les moyens à mettre en œuvre,
- ☞ Maîtriser le projet en mesurant son avancement et les écarts éventuels avec les engagements pris,
- ☞ S'assurer que la qualité définie est respectée.

# L'évolution des méthodes

## □ **Les premières méthodes d'analyse (années 70)**

Découpe fonctionnelle (fonctionnelle et hiérarchique) d'un système.

## □ **L'approche systémique (années 80)**

Modélisation des données + modélisation des traitements (Merise, Axial, ..).

## □ **L'émergence des méthodes objet (1990-1995)**

- ✓ Prise de conscience de l'importance d'une méthode spécifiquement objet : comment structurer un système sans centrer l'analyse uniquement sur les données ou uniquement sur les traitements (mais sur les deux) ?
- ✓ Plus de 50 méthodes objet sont apparues durant cette période (Booch, Classe-Relation, Fusion, HOOD, OMT, OOA, OOD, OOM, OOSE...) !
- ✓ Aucune méthode ne s'est réellement imposée.

# Des méthodes fonctionnelles aux méthodes objet

- **Les premiers consensus (1995)**
  - ✓ **OMT** (Object Modeling Technique - James Rumbaugh) - Méthode d'analyse et de conception orientée objet. Vues statiques, dynamiques et fonctionnelles d'un système.
  - ✓ **OOD** (Object Oriented Design - Grady Booch). Vues logiques et physiques du système.
  - ✓ **OOSE** (Object Oriented Software Engineering - Ivar Jacobson). Couvre tout le cycle de développement. Une des plus anciennes méthodes objet focalisée sur le modèle statistique.
- **L'unification et la normalisation des méthodes (1995-1997)**
  - ✓ UML (Unified Modeling Language) est né de la fusion de ces 3 méthodes qui ont le plus influencé la modélisation objet au milieu des années 90

Fin 1997, UML devient une norme **OMG** (Object Management Group).  
L'OMG est un organisme à but non lucratif, créé en 1989 à l'initiative de grandes sociétés (HP, Sun, Unisys, American Airlines, Philips...). Son rôle est de promouvoir des standards qui garantissent l'interopérabilité entre applications orientées objet, développées sur des réseaux hétérogènes.

# Présentation de la méthode Merise

# Vue d'ensemble de Merise

## 1. Vue d'ensemble de la démarche

Qu'est ce que Merise ?

Création : 1978-1979

**Plus qu'une méthode d'analyse informatique, une démarche de construction des Systèmes d'information.**

## A quoi sert Merise ?

- **En ce qui concerne les données :** A identifier le nombre et la nature des tables, les articulations et la ventilation des informations entre ces tables, afin que l'ensemble soit le plus efficace et évolutif possible,
- **Pour les traitements :** A identifier les fonctionnalités selon une approche "top / down" ("du général au particulier"), leur découpages et leurs enchaînements.

Merise est un travail d'anticipation : Elle sert à préparer les développements informatiques et à chiffrer (en coût, en temps et en énergie) ces chantiers, quelle que soit leur échelle.

# Les niveaux d'abstraction

**Il y a 3 niveaux d'abstraction :**

- 1. Le niveau conceptuel**
- 2. Le niveau organisationnel**
- 3. Le niveau physique**

**1. Le niveau conceptuel :**

Il consiste à répondre à la question QUOI ?

Quoi faire, avec quelles données ?

A ce niveau, on ne se préoccupe pas de l'organisation du travail ni du matériel utilisé.

**Les deux modèles sont le Modèle conceptuel des données (MCD) et le Modèle conceptuel des traitements (MCT).**

**2. Le niveau organisationnel :**

Il consiste à répondre à la question QUI ?, OU ?, QUAND ?

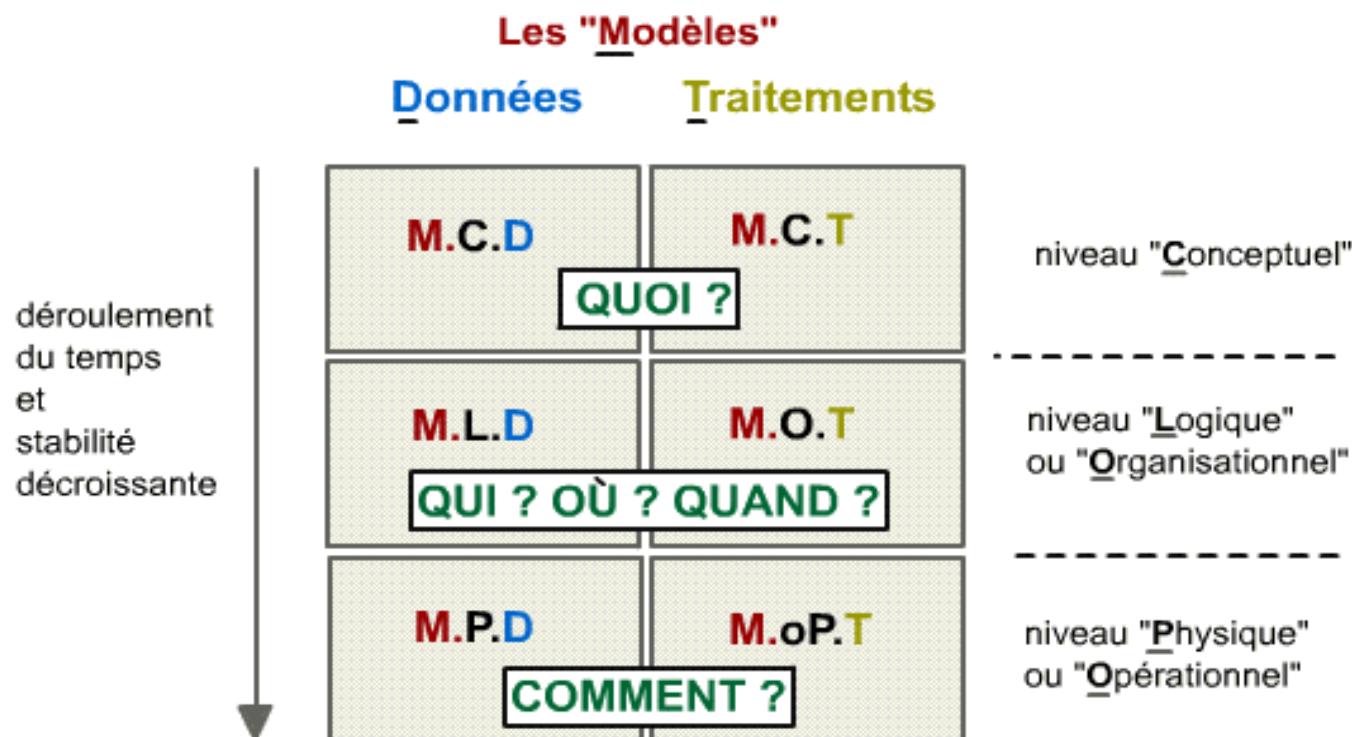
C'est à ce niveau que sont intégrés les critères d'organisation de travail.

On tient compte (ou on propose) des choix d'organisation de travail comme la répartition des traitements entre l'homme et la machine, le mode de fonctionnement (temps réel, temps différé).

**3. Le niveau physique**

L'organisation du stockage

# Dissociation des données et des traitements



# Déroulement des données / traitements

- **Le "niveau conceptuel"** (le "Quoi ?"), aboutissant aux M.C.D. ("Modèle conceptuel des données") et M.C.T. ("Modèle conceptuel des traitements"). A ce stade, données et traitements sont étudiés de manière parallèle, dissociée.
- **Le "niveau logique"** (pour les données) et le "niveau organisationnel" (pour les traitements) (le "Qui?", le "Quand?", le "Où?") correspondants aux M.O.T ("modèle organisationnel des traitements") et M.L.D. ("Modèle logique des données").
- **Le "niveau physique"** (pour les données) aboutissant à la création des tables, et le "niveau opérationnel" (pour les traitements) enclenchant analyse détaillée de chaque traitement, et développements.

# Le M.C.D (Modèle conceptuel de données)

# Démarche

Représentation statique, sous forme schématique, de la situation respective des données d'un domaine de gestion.

Ce schéma est conçu pour être très stable dans le temps.

**Son objectif :** définir (identifier) toutes les données utilisées, les regrouper en ensembles appelés entités, et de lier ces entités par des relations, dans un modèle définit et compréhensible par toute personne connaissant la "syntaxe" du MCD.

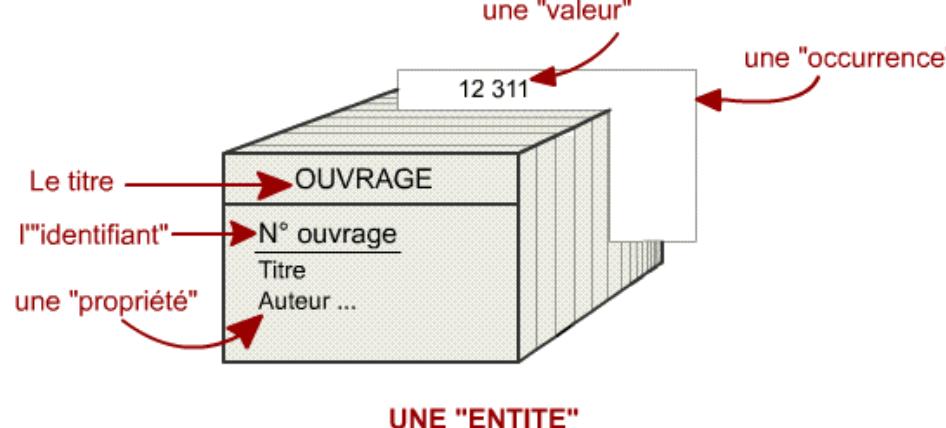
Le MCD regroupe les informations à traiter, le "quoi" du système.

**Les étapes du MCD :**

1. Catalogue des données
2. Épuration (polysèmes et synonymes)
3. Détermination des entités
4. Détermination et affectation des propriétés
5. Recensement des associations
6. Détermination des cardinalités

# Entité

Représentation d'un objet réel, ayant une existence et une raison d'être dans le système d'information.



# Principes d'une entité

- Une entité est pourvue d'une **existence propre** et est conforme aux choix de gestion de l'entreprise.
- Une entité peut être **un acteur** : client, usine, produit => pourvue d'une existence intrinsèque.
- Une entité peut être **un flux** : commande, livraison => existe par l'intermédiaire d'acteurs.

## Les propriétés :

Une propriété est une donnée élémentaire qui qualifie l'entité à laquelle elle se rapporte :

- Chaque propriété prend des valeurs qui sont appelées occurrences de la propriété,
- Chaque propriété a un domaine de définition (ensemble de valeurs possibles),
- Chaque propriété se rattache toujours à une entité.

## Identification d'une Entité :

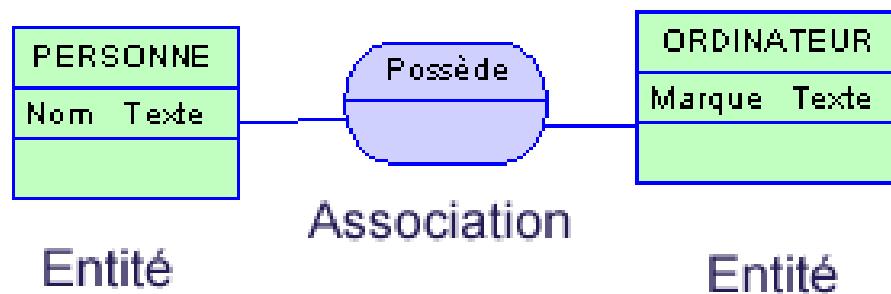
C'est une propriété (ou ensemble de propriétés) particulière qui permet d'identifier de façon unique une occurrence de l'entité.

- Pour être identifiant, la ou le groupe de propriétés ne doit pas prendre plusieurs fois la même valeur sur l'ensemble des occurrences de l'entité.
- L'identifiant figure en premier dans la liste des propriétés
- Il est souligné

# Association (ou Relation)

Objet permettant d'associer deux ou plusieurs entités. Ce lien est nommé et est, par convention, très souvent un verbe à l'infinitif.

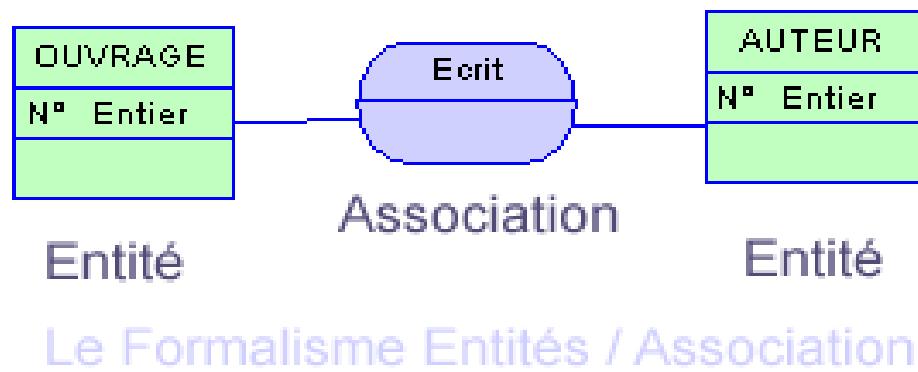
ex : entre deux entités, Personne et Ordinateur, une relation nommée Posséder peut être mise, et on lit "une personne possède un ordinateur" et, dans l'autre sens, 'un ordinateur est possédé par une personne".



Le Formaliste Entités / Association

# Association (ou Relation) - suite

- **ex** : entre deux entités, Ouvrage et Auteur, une relation nommée Écrire peut être mise, et on lit "un Auteur a écrit un Ouvrage" et, dans l'autre sens, 'un Ouvrage est écrit par un Auteur".



# Association (ou relation) - fin

- **Définition**

- Lien « sémantique » reliant des entités et présentant un intérêt pour l'entreprise.

- **Association porteuse**

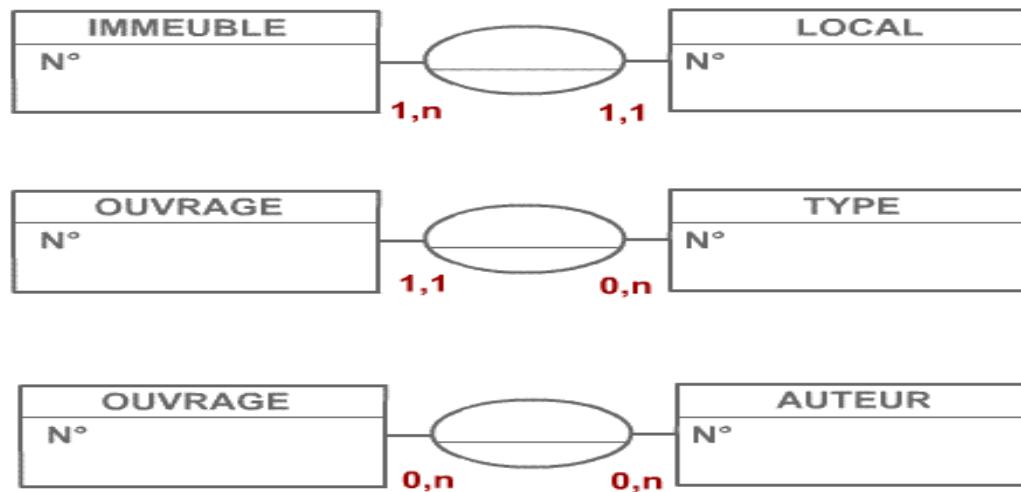
- Association qui porte des propriétés.

- **Dimension d'une association**

- Association réflexive : lien de l'entité sur elle-même
- Association binaire : lien entre deux entités
- Association ternaire : lien entre trois entités
- Association n-aire : lien entre n entités

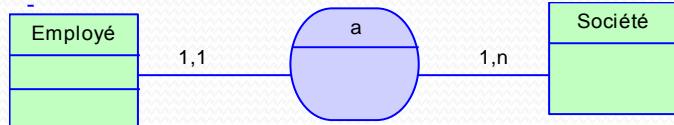
# Cardinalité d'une Association

- C'est le nombre d'occurrences, minimal et maximal, d'une association par rapport à chaque occurrence d'une entité donnée. D'une entité donnée vers une association donnée.



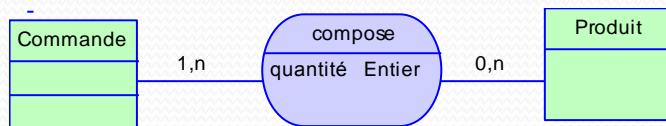
**LES CARDINALITES : Quelques exemples**

# Exemples de cardinalité



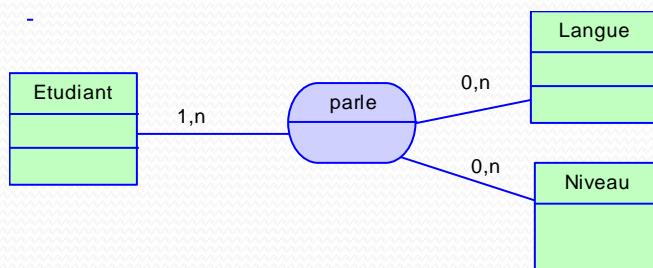
Un employé a une et une seule société. Une société a 1 ou n employés.

Ex 2 :



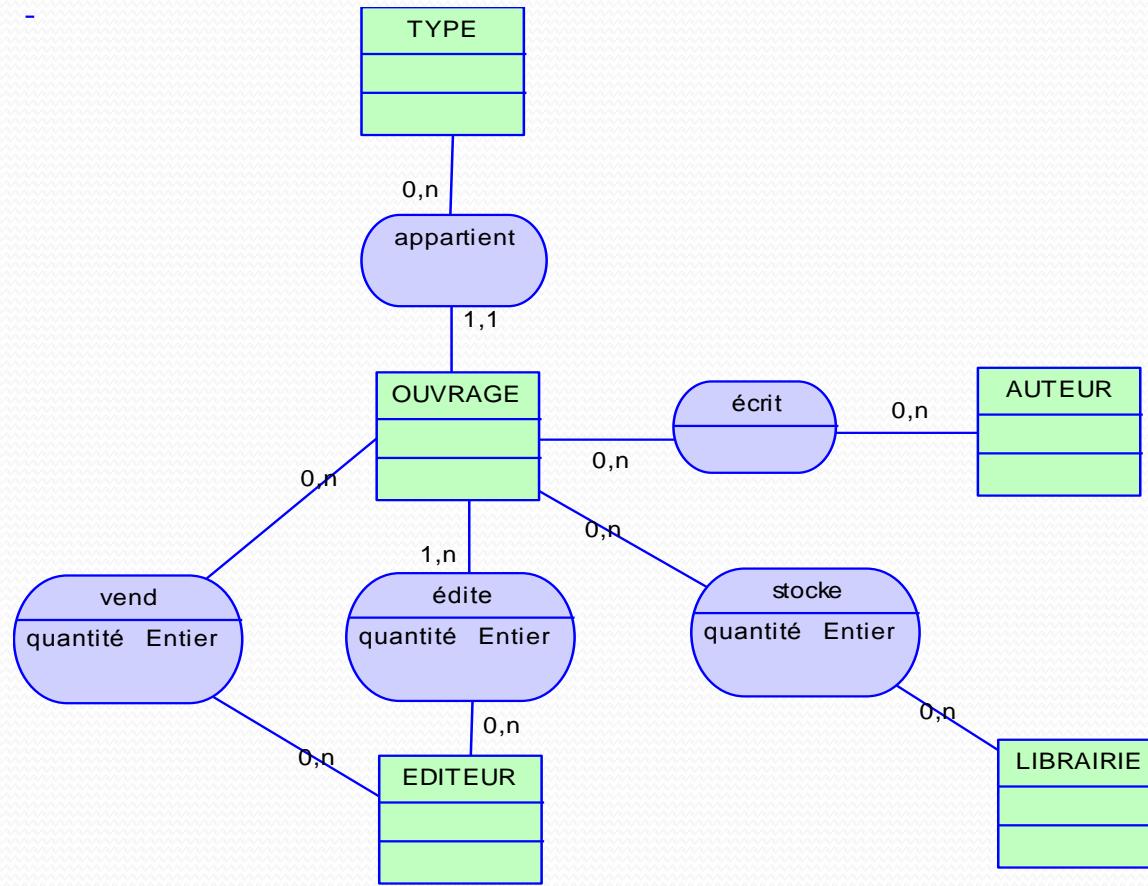
Une commande est composée de 1 ou n produits distincts en certaine quantité. Un produit est présent dans 0 ou n commandes en certaine quantité.

Ex 3 :



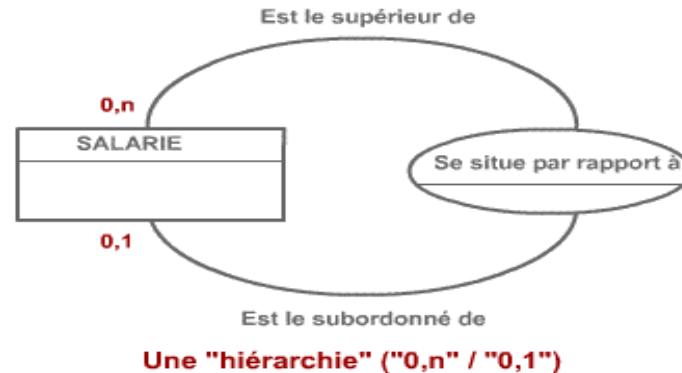
Un étudiant parle une ou plusieurs langues avec un niveau. Chaque langue est donc parlée par 0 ou n étudiants avec un niveau. Pour chaque niveau, il y a 0 ou plusieurs étudiants qui parlent une langue.

# Une centrale d'achat : les cardinalités

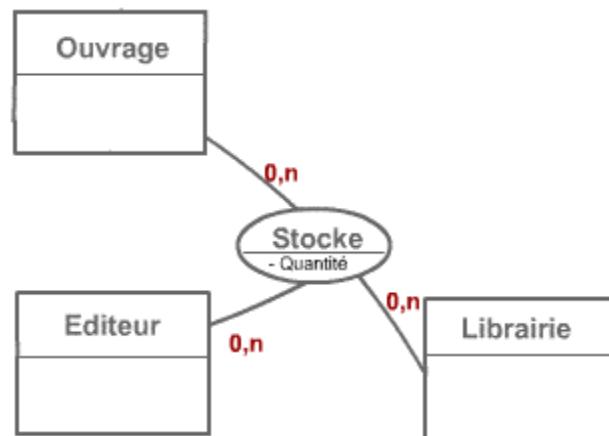


# Association réflexive et ternaire

Cas des associations de dimension "1" (dites "réflexives") :



Cas des associations de dimension "3"



# Première étude de cas

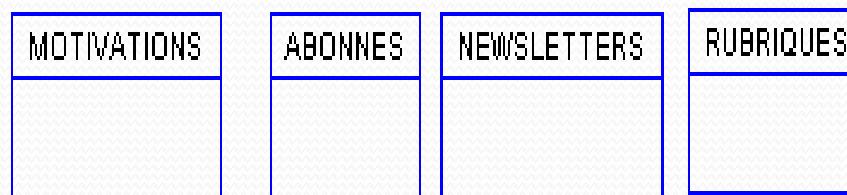
# Etude de cas 1

- Le Système d'Information se décrit ainsi :

*"Un abonné est inscrit à une ou plusieurs rubrique. Chaque rubrique envoie une NewsLetter chaque semaine aux abonnés de la rubrique correspondant. Un abonné a une motivation d'inscription parmi plusieurs possibles."*

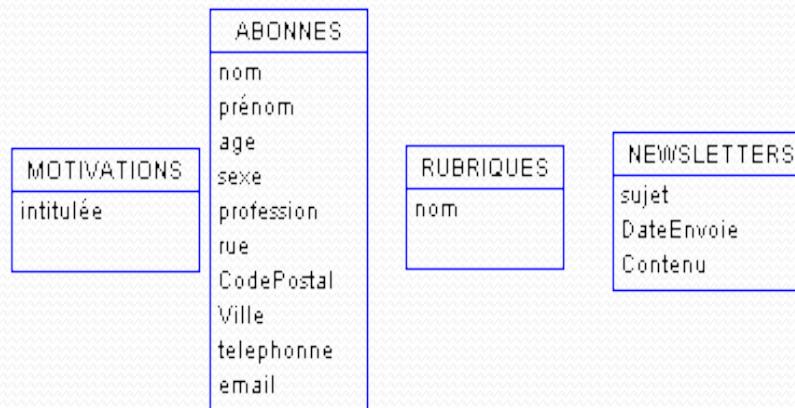
# Identifier les entités présentes

- L'entité ABONNES représente l'ensemble des abonnés. L'entité RUBRIQUES l'ensemble des rubriques auxquelles l'abonné peut s'inscrire. L'entité NEWSLETTERS représente les newsletters envoyées, MOTIVATIONS l'ensemble des motivations d'inscription des abonnés.
- Généralement, une entité est créée dans le système d'information si elle possède au moins 2 occurrences. Chaque élément d'une entité est appelé une **occurrence** de l'entité.



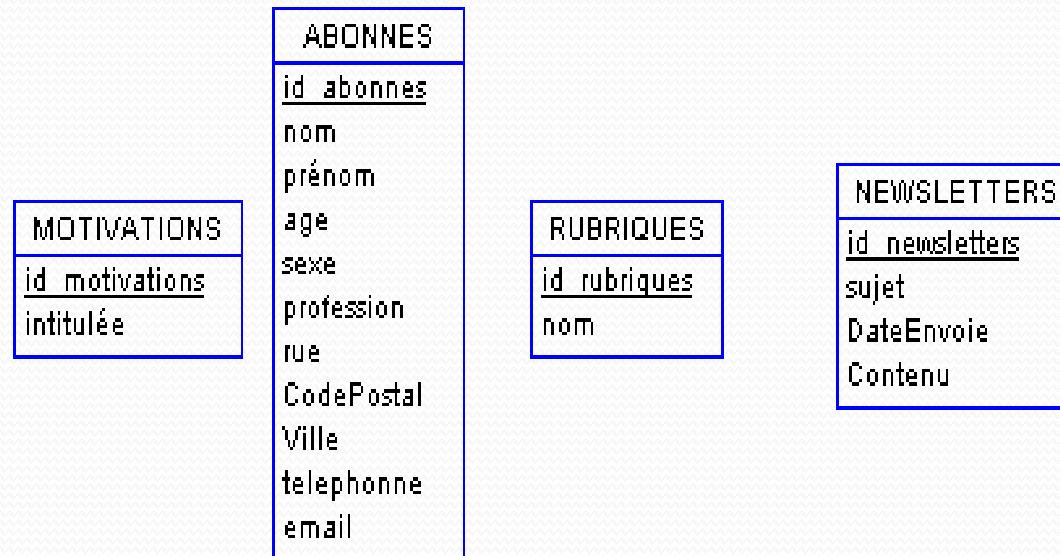
# Lister les propriétés des entités

- Afin de ne pas en avoir trop, on se limite généralement aux propriétés nécessaires au développement.
- Chaque propriété doit avoir une seule valeur possible pour chaque occurrence, sinon il s'agit d'une entité. Elle doit de plus être élémentaire et non-décomposable.



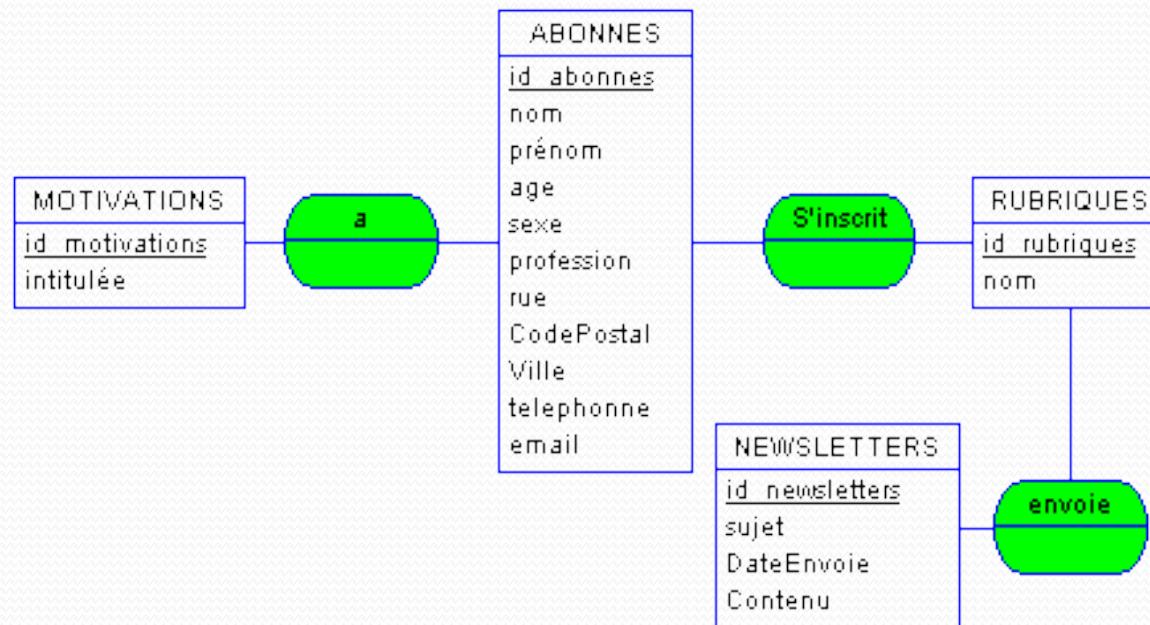
# Identifier de manière unique chaque occurrence

- L'**identifiant** de l'entité peut être une référence interne, un code, ou plus généralement un nombre entier. Cette propriété est soulignée afin de mettre en évidence son rôle d'identifiant.

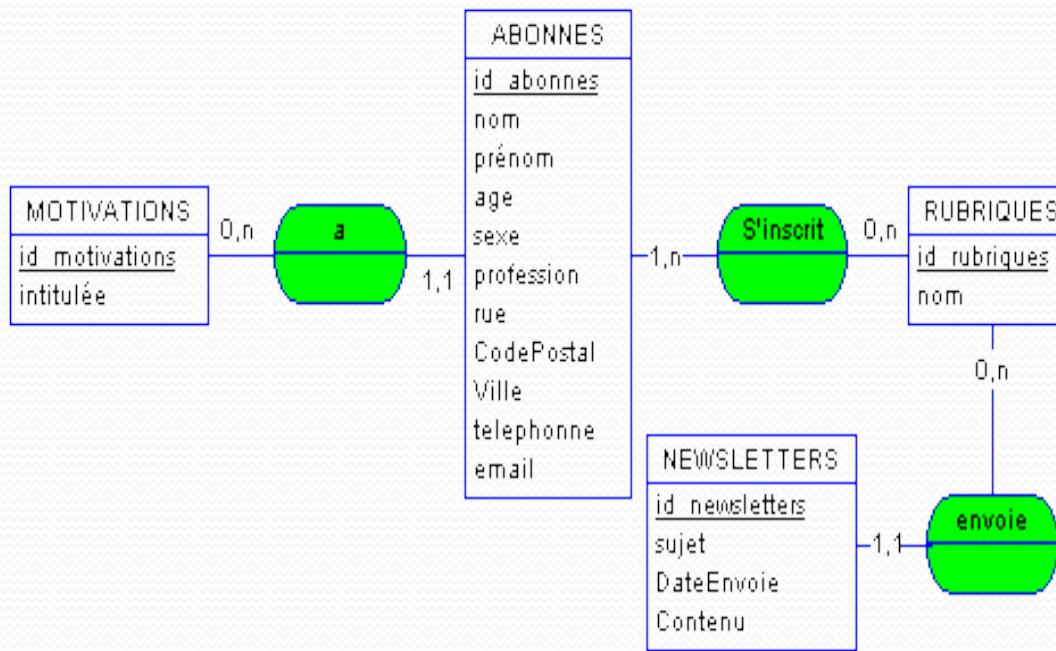


# Etablir les relations entre les différentes entités

- Généralement, la simple transposition du texte suffit, les sujets et compléments d'objets étant les entités, et les verbes les relations.



# Identifier les cardinalités



# Deuxième étude de cas avec Merise

# Etude de cas 2

« Une entreprise X vend des véhicules toutes marques qu'elle stocke dans de grands entrepôts. Dans un même entrepôt, nous pouvons trouver plusieurs marques de véhicules, cependant, pour des raisons de logistiques, le gérant de la société X a exigé de ses employés qu'une marque ne puisse se trouver que dans un seul entrepôt. Chaque attaché commercial gère son propre portefeuille de clients. L'entreprise X souhaite établir des statistiques commerciales sur ses ventes de véhicules : nombre de véhicules achetés par un client, chiffre d'affaire réalisé par une marque, mais aussi sur les marques entreposées dans un entrepôt. »

**Objet :** Vente de véhicules toute marque

**Application :** Statistiques commerciales

**Résultats attendus :**

- ✓ Nombre de véhicules achetés par un client ?
- ✓ Chiffre d'affaire réalisé par une marque ?
- ✓ Quelles sont les marques entreposées dans un entrepôt ?

# La recherche des données

- **Dans les interviews**
  - Les données peuvent être éparpillées dans le discours.
- **Une donnée est caractérisée par :**
  - Un nom
  - Une définition (explique à quoi elle sert)
  - Un domaine de définition
  - Une provenance
  - Un mode de calcul ( si donnée calculée )
  - Une décomposition ( si donnée non atomique )

## **Données :**

- ✓ Nom de marque
- ✓ Nom de dépôt
- ✓ Nom du type
- ✓ Puissance fiscale
- ✓ Nom du responsable commercial pour une marque
- ✓ Prix unitaire d'un type de véhicule
- ✓ Adresse de dépôt
- ✓ Nom, adresse du client
- ✓ Quantité d'une vente
- ✓ Date d'une vente
- ✓ Nom de l'attaché commercial
- ✓ Adresse de l'attaché commercial

## **Contraintes sur les données :**

- ✓ Un dépôt peut-être multi-marque,
- ✓ Une marque ne se trouve que dans un seul entrepôt,
- ✓ Un attaché gère plusieurs clients,
- ✓ Un client est géré par un seul attaché

# Regroupement en entités

- Les données sont explicitement des propriétés d'entités nommées.
- Les données sont regroupable dans une entité non explicite, mais sémantiquement identifiable.
- Les données sont isolées, il faut construire des entités supplémentaires.

# Comment nomme-t-on une entité ?

« Une entité est représentée par un nom qui sert de discriminant principal (existentiel) pour un objet. »

C'est sur la base de ce nom que l'on peut dire qu'une occurrence (ligne) est ou n'est pas une instance de cette entité.

Une entité désigne une « classe », une « collection » de choses

## **1. Repérer les entités :**

Les entités sont les objets de gestion essentiels du système d'information.

L'entité est une ensemble dont chaque élément est un élément particulier.

Dans notre exemple, que gère t-on ?

En première lecture, 3 entités ressortent :

1. Les types de véhicule,
2. Les clients,
3. Les dépôts.

## **2. Attribuer à chaque entité son identifiant et ses propriétés :**

Chacune de ces entités possède des caractéristiques, qui sont appelées propriétés :

- Une propriété ne doit figurer qu'à un seul endroit du modèle,
- Elle doit être significative

Parmi ces propriétés, il peut y en avoir une ou plusieurs qui permettent de définir sans ambiguïté un élément parmi l'ensemble des éléments : **l'identifiant**.

L'identifiant sert à désigner sans ambiguïté une occurrence de l'entité.

Comment allons-nous construire l'entité TYPE DE VEHICULE ?

**Identifiant : *Nom du type***

**Propriétés : *Prix***

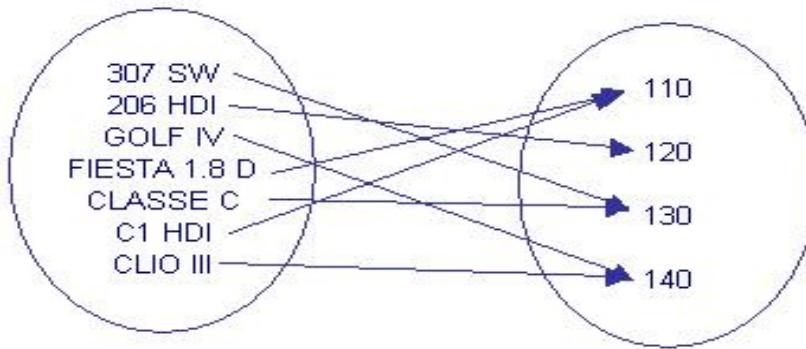
*Puissance fiscale*

*Nom de marque*

## 2. Attribuer à chaque entité son identifiant et ses propriétés :

Il faut alors vérifier qu'à toute valeur prise par l'identifiant ne correspond qu'une valeur de chaque propriété. Nous appelons cette règle : *la règle d'énumération*.

Ici, pour une valeur prise par un nom de type (identifiant), nous n'avons qu'une valeur de puissance fiscale.



Constituons l'entité DEPOT :

**Identifiant :** Nom de dépôt

**Propriétés :** *Adresse de dépôt*

Pour l'entité CLIENT, il est légitime de modéliser :

**Identifiant :** Code client

**Propriétés :** *Nom du client*

*Adresse du client*

*Nom attaché commercial*

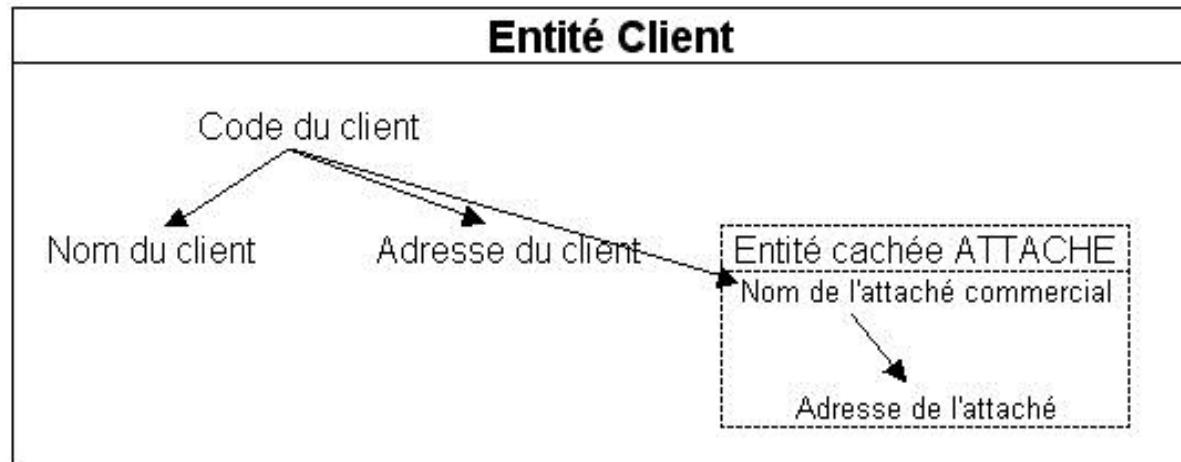
*Adresse attaché*

## 2. Attribuer à chaque entité son identifiant et ses propriétés :

Il faut ensuite vérifier toutes les propriétés d'une entité dépendent directement de l'identifiant. Nous appelons cette règle : *la règle de dépendance directe*.

Pour l'entité CLIENT, il convient donc de se poser la question suivante :

*L'adresse de l'attaché commercial est-elle une propriété du client ou de l'attaché commercial ?*



Il convient donc de modéliser ainsi :

Entité : CLIENT

ATTACHE

Identifiant : Code Client

Nom de l'attaché commercial

Propriétés : Nom du client

Adresse de l'attaché commercial

Adresse du client

## 2. Attribuer à chaque entité son identifiant et ses propriétés :

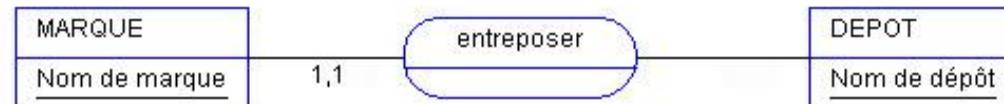
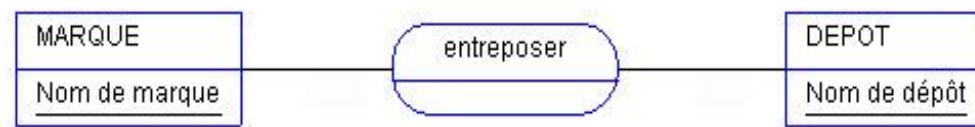
Pour l'entité TYPE DE VEHICULE, nous avons défini une propriété Nom de marque. Cependant, le responsable commercial est dépendant de la marque. MARQUE est donc une entité cachée.

Il convient donc de modéliser ainsi :

Entité :	MARQUE	TYPE DE VEHICULE
Identifiant :	<u>Nom de marque</u>	<u>Nom du type</u>
Propriétés :	<i>Responsable commercial</i>	<i>Prix</i> <i>Puissance fiscale</i>

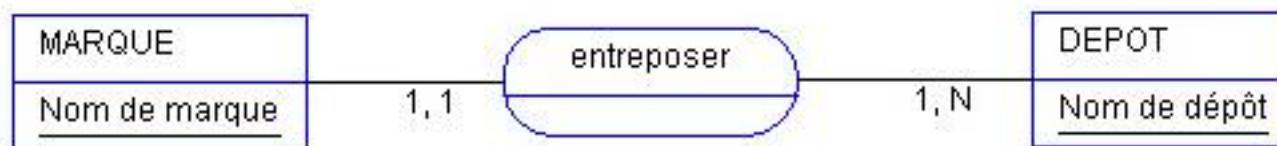
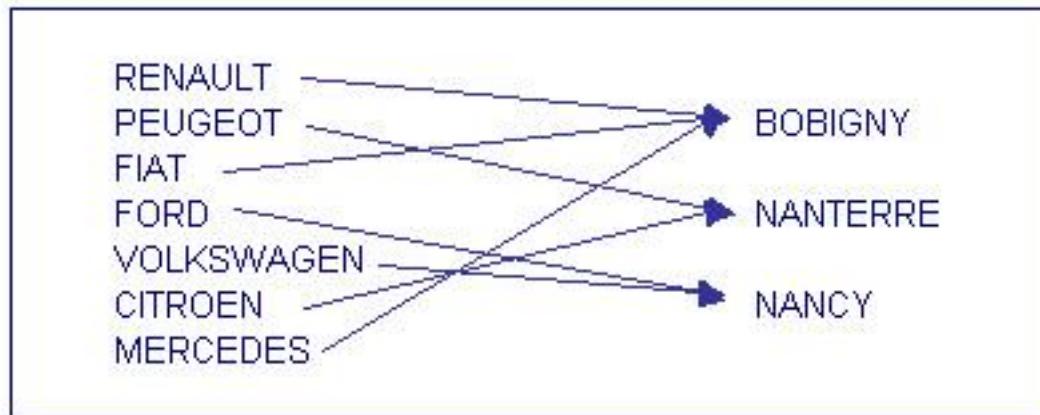
## 3. Définition des relations entre les Entités et cardinalité :

- Relation entre MARQUE et DEPOT :



### 3. Définition des relations entre les Entités et cardinalité :

- Relation entre MARQUE et DEPOT :



(1,1) : Une marque est entreposée dans un seul entrepôt.

(1,N) : Dans un entrepôt sont entreposées une ou plusieurs marques.

### 3. Dépendances fonctionnelles entre Entités (DF) :

La relation qui lie un TYPE DE VEHICULE à une MARQUE est «appartenir». Il s'agit d'une relation hiérarchique : à un type de véhicule donné ne correspond qu'une seule marque, et à une marque correspondent plusieurs types de véhicules. Ici, TYPE DE VEHICULE détermine totalement MARQUE.

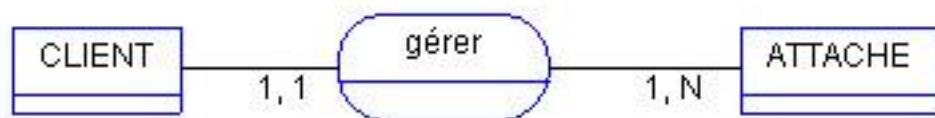
Nous appelons cette relation fonctionnelle (DF), et elle se note de la manière suivante :

**TYPE DE VEHICULE -> MARQUE**

**Qu'en est-il de ATTACHE COMMERCIAL et de CLIENT ?**

**CLIENT -> ATTACHE COMMERCIAL**

(A un client ne correspond qu'un attaché commercial)



#### 4. Relations entre 3 Entités :

Regardons à présent le problème des quantités élémentaires de ventes.

Cette données est une propriété de la relation « vendre », liant CLIENT et TYPE DE VEHICULE : la quantité ne prend de sens que si l'on connaît conjointement le client et le type.

Les cardinalités se lisent alors :

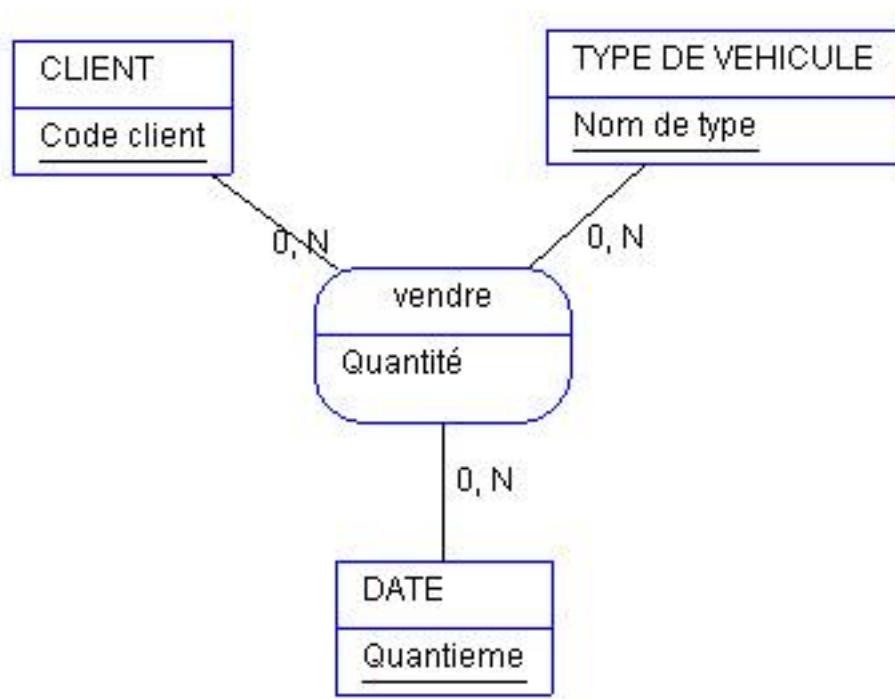
- Pour un **type de véhicule**, il peut y avoir de 0 à N relations « vendre », autrement dit il peut y avoir plusieurs ventes pour un même type de véhicule,

Pour un **client**, il peut y avoir 0 à N relations « vendre », en d'autres termes, il peut y avoir plusieurs ventes pour un même client.

A une **date**, il peut y avoir 0 à N relations « vendre ». Ce qui se traduit par : certains jours, il y a plusieurs ventes.

A une occurrence de la relation « vendre »

correspond un triplet (MARQUE, CLIENT, DATE) et un seul.



## 5. Vérifier la règle de pleine dépendance :

Les propriétés d'un relation doivent dépendre de la totalité des entités qu'elle relie.

- ⇒ Pour chaque propriété d'une relation, toutes les entités liées par la relation doivent servir à son identification.

### Exemple :

Supposons que nous ayons à gérer des taux de remise dépendant à la fois du **TYPE DE VEHICULE** et du **CLIENT**, il serait erroné de faire figurer ce taux de remise dans la relation « vendre », puisque ce taux ne dépend pas de **DATE**. Il conviendrait dans ce cas de créer une nouvelle relation « a pour remise ».

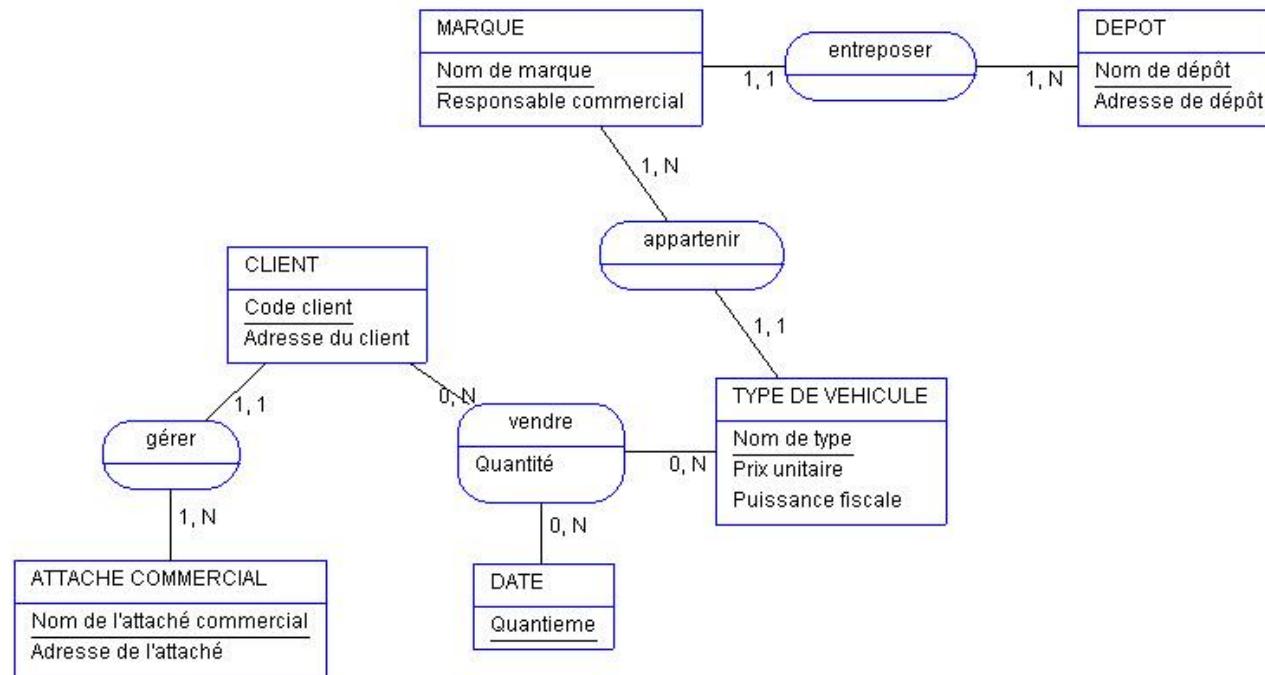
De plus, on serait conduit à une grande redondance d'information, puisque l'on stockerait plusieurs fois un même taux de remise.

## 6. Synthèse et compléments :

1. Vérifier si le modèle est bien construit :
  - a. Contrôler l'ensemble du modèle en vérifiant que chaque propriété se trouve en un seul endroit du modèle.
  - b. Contrôler chaque Entité en vérifiant :
    - Que chaque Entité possède un identifiant,
    - Que chaque propriété est significative,
    - La règle d'énumération,
    - La règle de dépendance directe

## 6. Synthèse et compléments :

- c. Contrôler chaque relation en vérifiant :
  - Q'une occurrence de relation ne lie qu'une et une seule occurrence de chacune des Entités qu'elle relie,
  - La règle d'énumération
  - Que chaque propriétés portée par une relation dépend de la totalité des entités qu'elle relie (règle de dépendance).
2. Vérifier si le modèle est capable de produire les résultats attendus.



# Normalisation du modèle

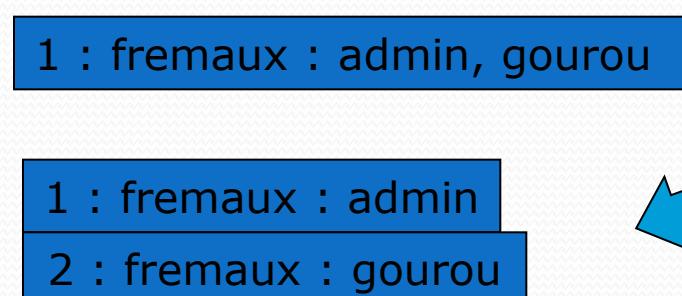
- But
  - Rendre le modèle le « plus propre possible ».
  - Limiter la redondance de données.
  - => Augmenter la fiabilité et diminuer la maintenance.
- Les règles de normalisation
  - Cinq formes normales.
  - Elles définissent la façon d'utiliser les entités, les attributs ou certaines contraintes sur les dépendances.

# Première forme normale

- Un identifiant
- Toutes les propriétés portent des valeurs atomiques, scalaires et non vectorielles)

~~utilisateur~~

<u>id</u>	nom	roles
1 : fremaux	admin, gourou	



utilisateur

<u>id</u>	nom	role
1	fremaux	admin

Un identifiant.

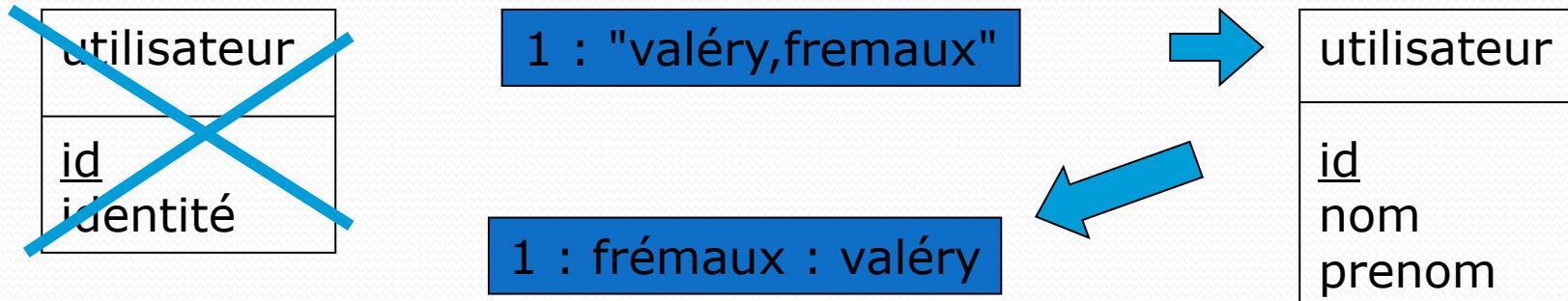
La propriété *roles* sous-entend que l'on peut mettre une liste de rôles

Un identifiant et des propriétés.

On impose à la propriété rôle de n'avoir qu'une seule valeur. On utiliserait plusieurs enregistrements.

# Première forme normale

- Un identifiant
- Toutes les propriétés portent des valeurs atomiques, scalaires et non vectorielles)

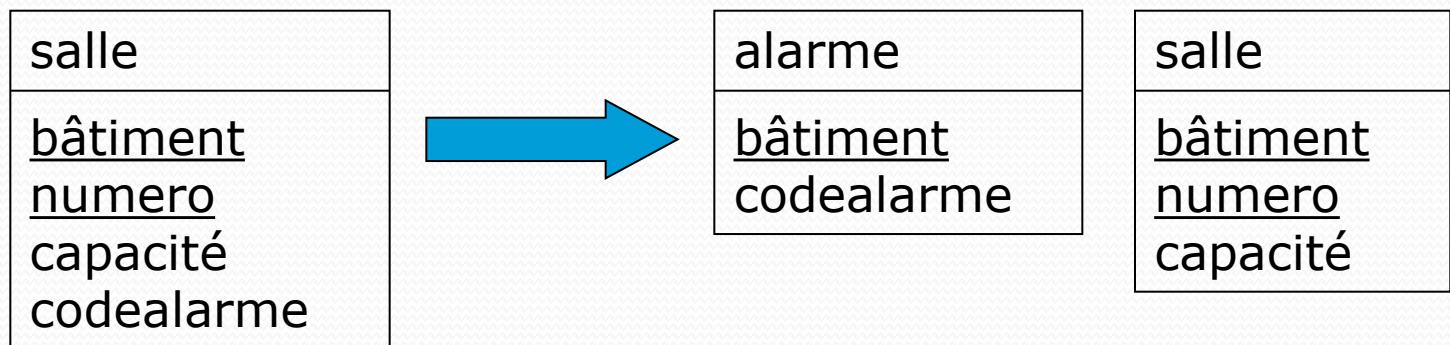


Un identifiant.  
La propriété *identité* sous-entend  
un agrégat d'informations  
distinctes que l'on « cache » dans  
un format scalaire (String)

Un identifiant et des propriétés.  
On remplace un attribut  
« composite » par des attributs  
scalaires.

# Deuxième forme normale

- On doit partir d'une 1ère Forme Normale (attributs atomiques et scalaires)
- On ne peut ajouter QUE des dépendances fonctionnelles complètes sur la clef (composite ou pas)

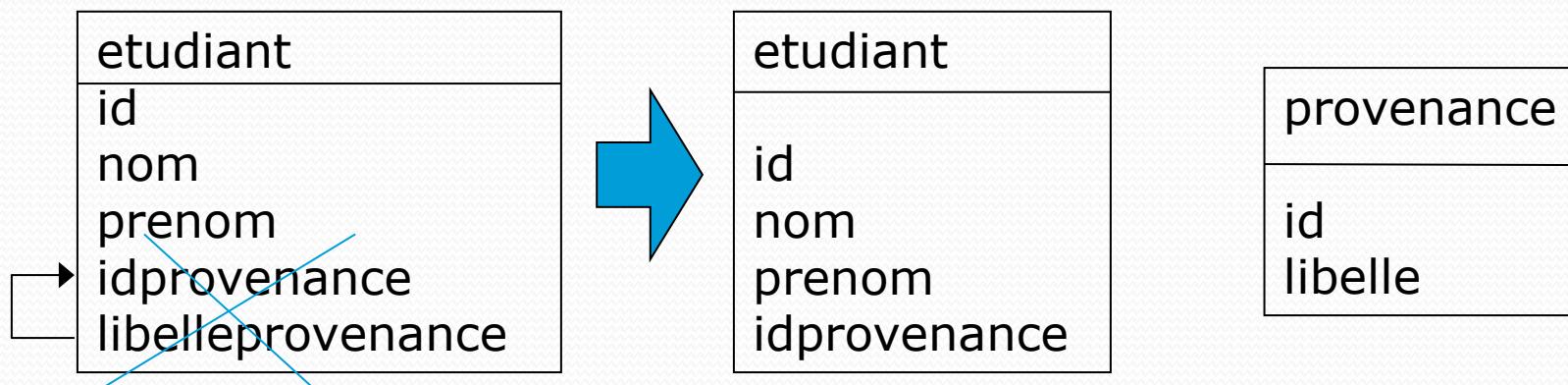


N : 40CD

N : 203 : 40

# Troisième forme normale

- On doit partir d'une Deuxième Forme Normale
- On interdit toute dépendance d'un attribut non identifiant sur un autre attribut non identifiant



**provenance.id => libelleprovenance ?**  
**etudiant.id => libelleprovenance ?**



**LibelleProvenance ne dépend pas directement de l'identifiant de table, mais d 'un autre attribut**

# Le M.L.D (Modèle Logique de Données)

# Passage à la bdd

- Comment passer du MCD à la BDD ?
- L'objet TABLE
- Comment créer un objet TABLE
- Les contraintes :
  - clé primaire
  - clé étrangère

# DU MCD à la base de données

Modèle conceptuel  
de données  
(MCD)

Diag. De  
Classe

On peut appliquer  
des règles de  
conversion

Modèle Logique de Données  
(MLD)

Transcription en langage SQL

Script de création des tables  
(BDD)

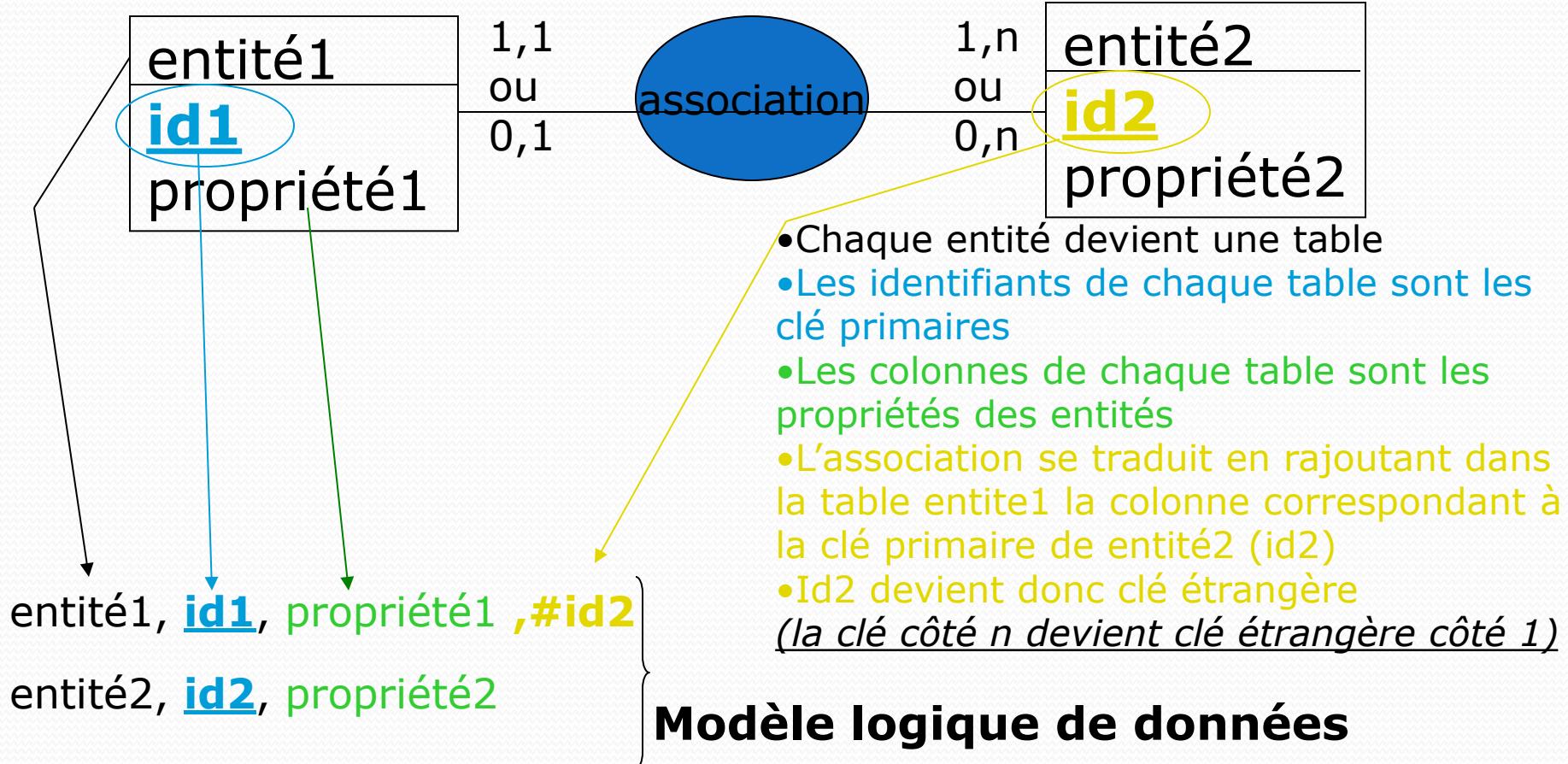
# Le MLD: les règles

- Règle 1 : Toute entité est représentée par une table.
- Règle 2 : Toute association qui associe plus de deux entités (ternaire et au-delà) est représentée par une table.
- Règle 3 : Toute association qui porte des données propres, c'est-à-dire indépendantes des entités devient une table.

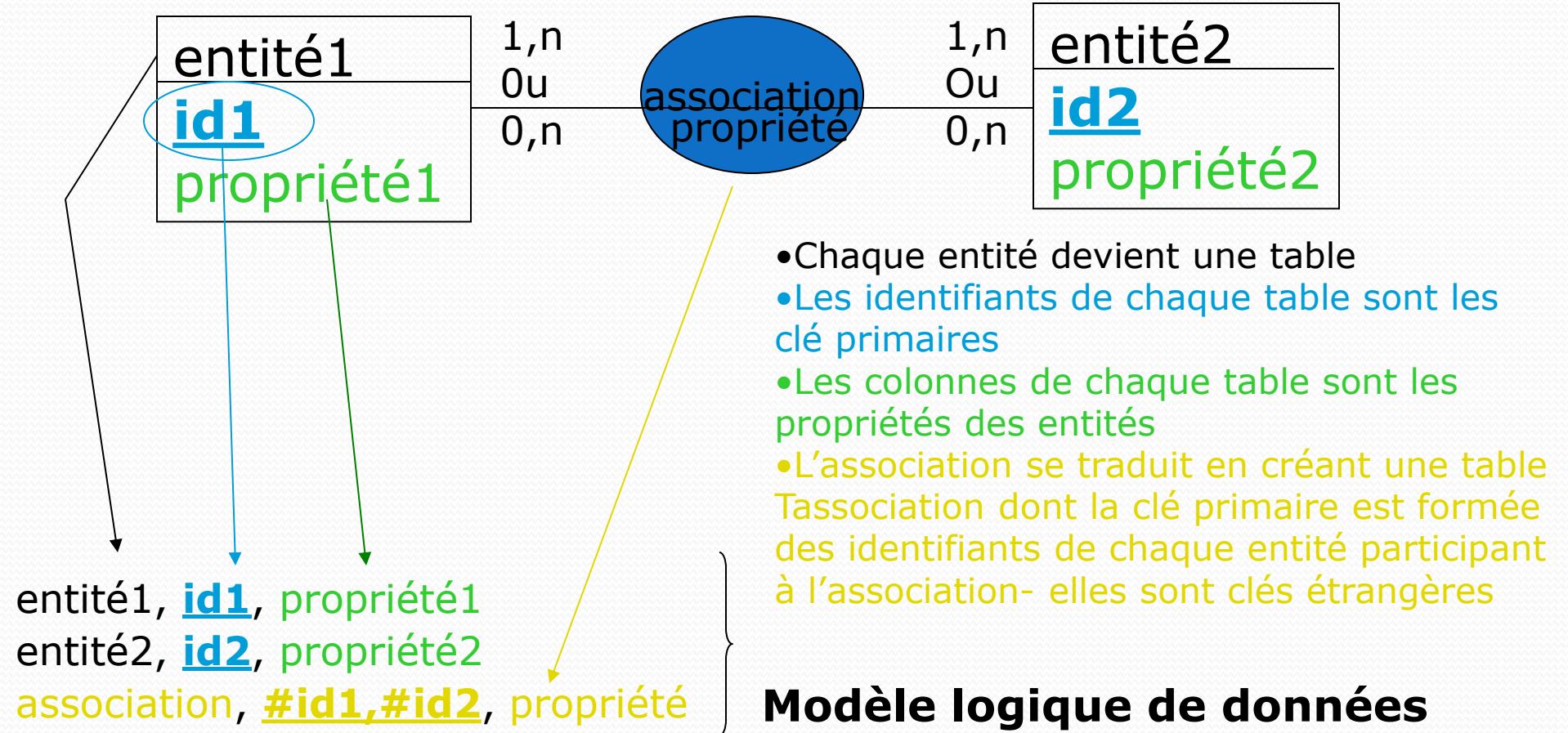
# Les règles à appliquer

- Règle 4 : Toute association binaire dont les cardinalités maximales sont n de chaque côté est une table (table dont les colonnes sont les attributs clefs des entités qu'elle relie ainsi que les éventuels attributs propres à l'association).
- Règle 5 : Une association de type père/fils, cardinalité maximum à n d'un côté et à 1 de l'autre, n'est pas représentée par une table. On indique les attributs clefs de l'entité père (côté (.,n)) dans le fils (côté (.,1)).

# Exemple : « relation père-fils »

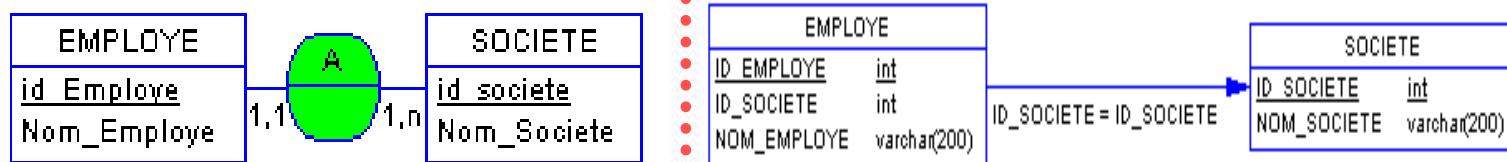


# Exemple : « relation matricielle » ou n..n

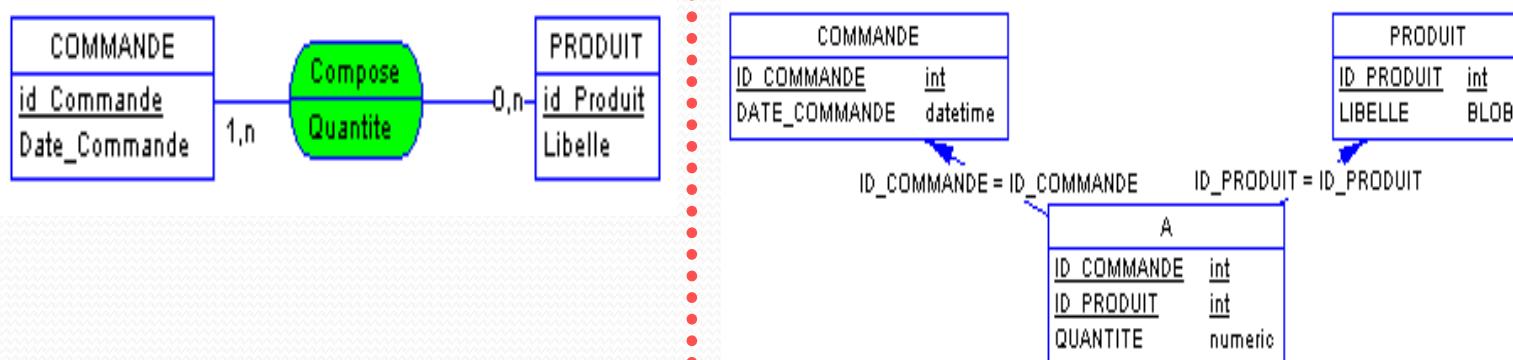


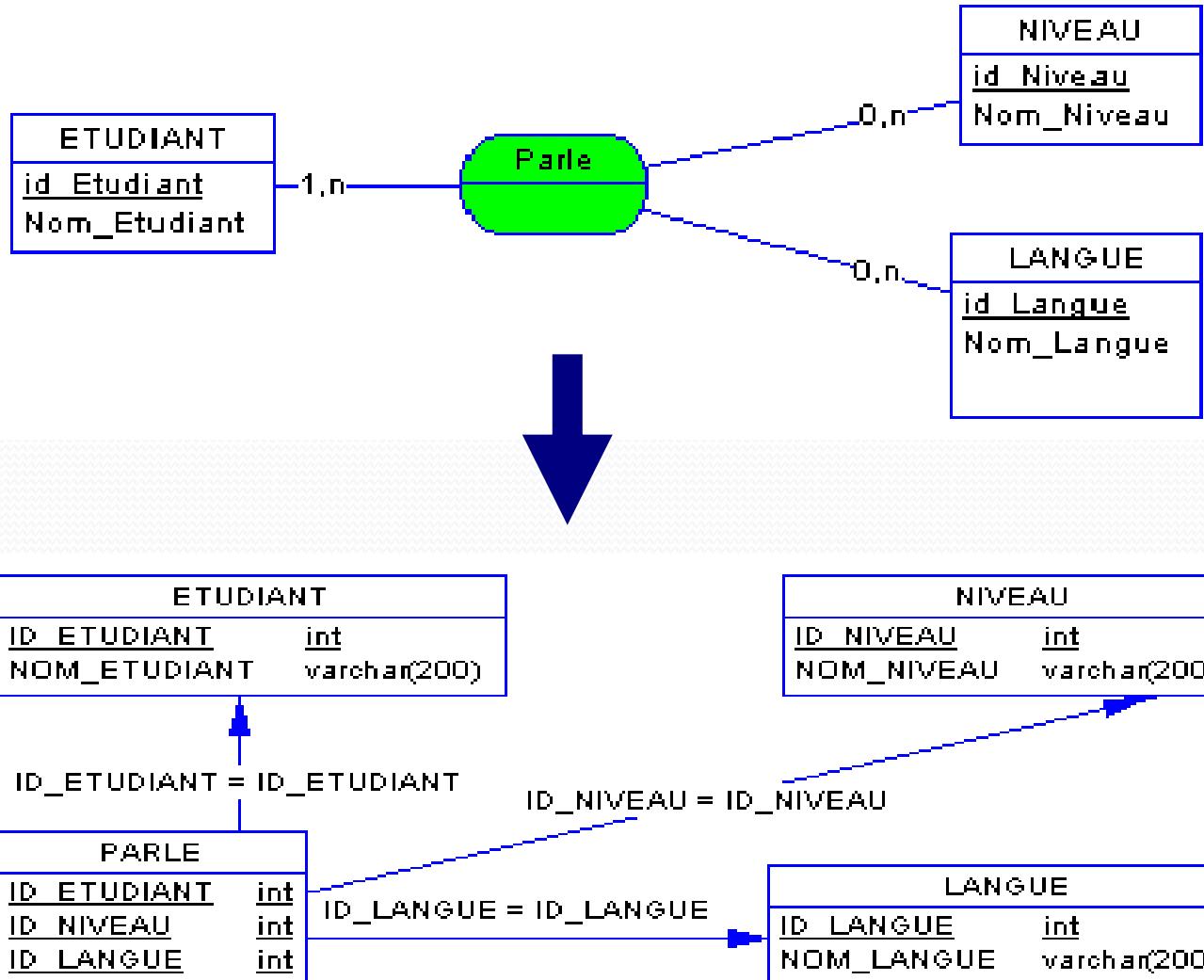
# Exemple

1er cas :

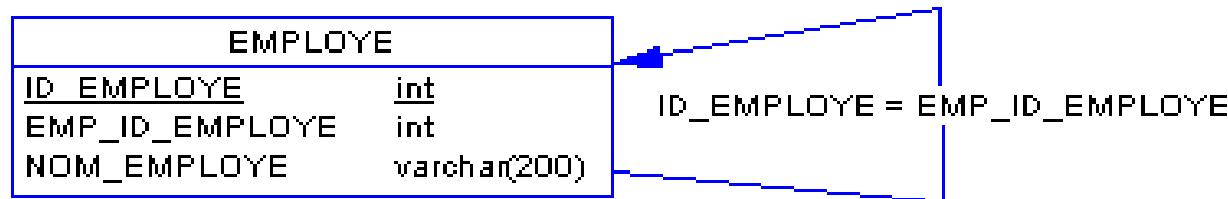
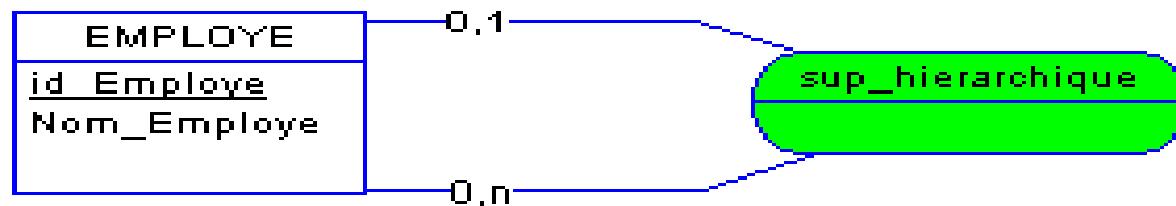


2ème cas :

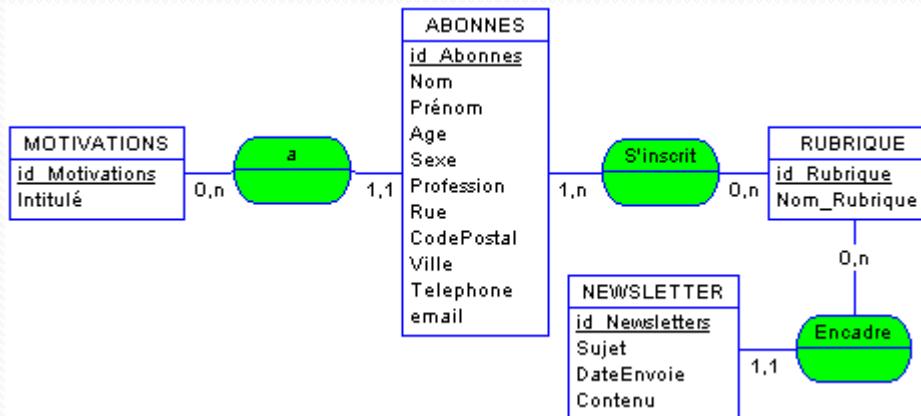




### 3eme cas :



# Etude de cas



MOTIVATIONS ( id\_Motivation, Intitule )

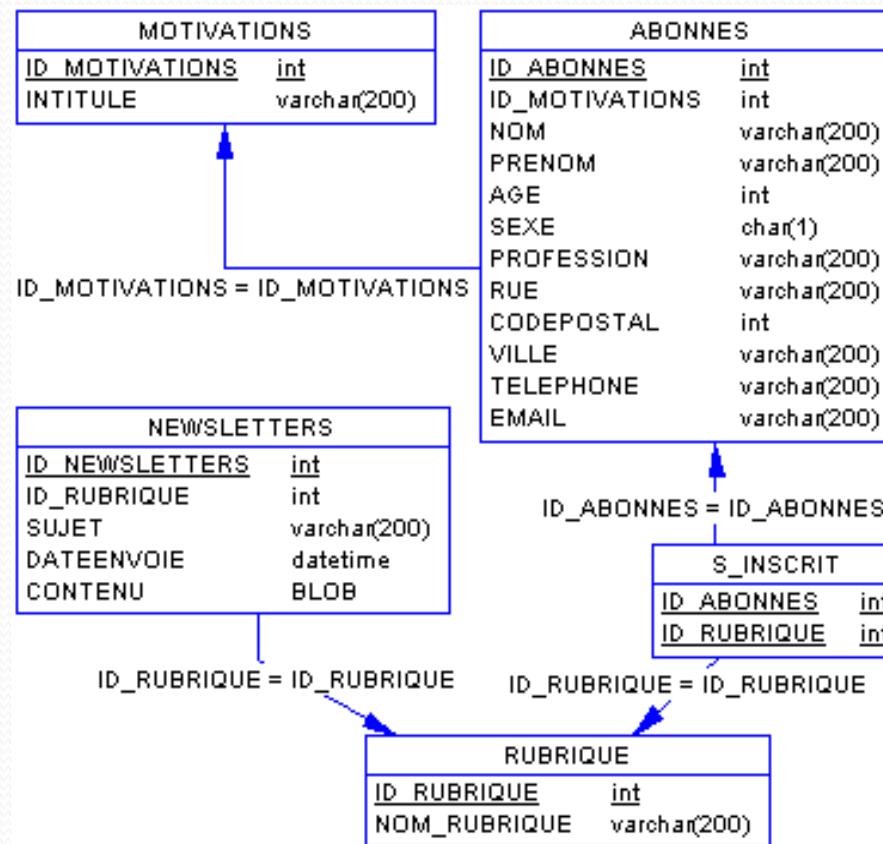
ABONNES ( id\_Abonnes, #id\_Motivation, Nom, Prenom, Age, Sexe, Profession, Rue, CodePostal, Ville, Telephone, Email )

S\_INSCRIT ( #id\_Abonne, #id\_Rubrique )

RUBRIQUE ( id\_Rubrique, Nom\_Rubrique )

NEWSLETTERS ( id\_Newsletters, #id\_Rubrique, Sujet, DateEnvoie, Contenu )

# Modèle physique



# Troisième étude de cas

## Merise

# Etude de cas 3

- *Système d'information :*
- *L'entreprise "WebCash" de vente par correspondance désire ajouter à son site un système de facturation visible en ligne pour ses clients. Chaque client, après authentification, pourra accéder à toutes les factures le concernant, qu'elles soient anciennes ou en cours de traitement indifféremment. Pour être sûr de bien se faire comprendre, "WebCash" fournit une facture type en disant : "C'est ça qu'on veut sur l'écran !"*

# Copie de la facture

**WebCash S.A.R.L**  
24, Avenue des Rêves roses  
75008 PARIS

**FACTURE N° 12345**

Paris, le 15/10/2000

**Nom :** BIDOCH  
**Prénom :** Robert  
**Adresse :** 12, rue du centre  
**Code Postal :** 70000  
**Ville :** Gray

<b>N° Article</b>	<b>Libellé</b>	<b>Prix Unitaire</b>	<b>Quantité</b>	<b>Prix</b>
234	Stylo Plume	12.5 F	1	12.50 F
568	Couteau Suisse	75.00 F	2	150 F
132	Serviette	30.00 F	1	30.00 F

**TOTAL TTC : 192.50 F**  
**Dont TVA 19.6% : 37.73 F**  
**A PAYER : 192.50 F**

Avec nos plus cordiaux remerciements

# Identification des entités

- **CLIENT** est l'ensemble des clients de la société WebCash. Une occurrence de cette entité est présentée par Robert BIDOCH, qui est le client à qui cette facture est destinée.
- **FACTURE** est l'ensemble des factures émises par WebCash, dont une occurrence est présente en "FACTURE N° 12345".
- **ARTICLE** est l'ensemble des articles vendus par WebCash, dont trois occurrences sont présentes, dénommés Stylo Plume, Couteau Suisse et Serviette.
- Une facture étant composée de plusieurs lignes, il aurait été possible de relever l'entité LIGNE\_FACTURE : elle n'est utile que si l'on désire archiver pour chaque ligne son Numéro. La base déduite aurait été sensiblement la même, démontrant ainsi que plusieurs solutions sont parfois possibles.
- La TVA est ici considérée comme constante et unique. Dans le cas contraire, elle aurait représenté l'entité TVA.
- La société WebCash ne représente pas une occurrence d'une entité, car c'est la seule société émettrice de facture de notre analyse.

# Lister les propriétés des entités

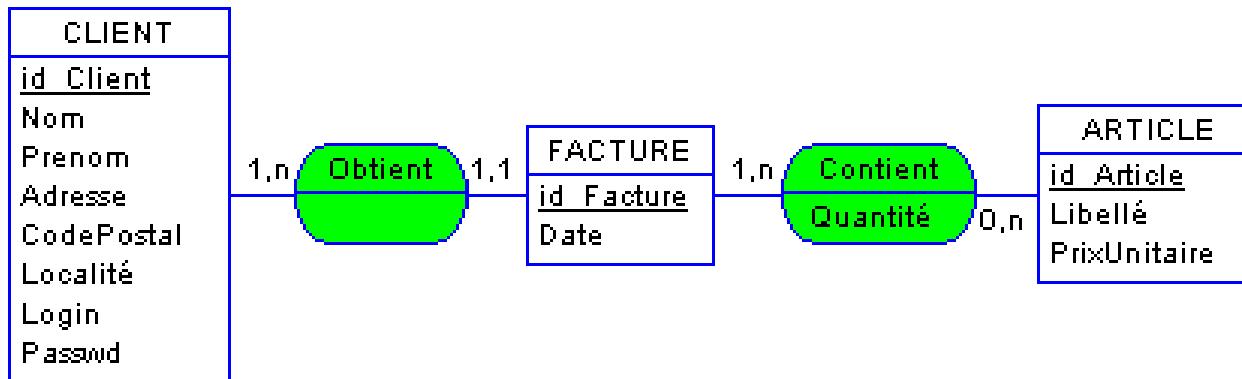
- Un **CLIENT** est caractérisé par son Nom, son Prénom, son Adresse, son CodePostal et sa Localité. Afin de pouvoir s'authentifier, il est aussi caractérisé par un Login et un Passwd.
- Une **FACTURE** est caractérisée par son Numéro, et sa Date d'émission.
- Un **ARTICLE** est caractérisé par son Numéro, son libellé, et son PrixUnitaire. Le prix total, de par son PrixUnitaire et sa quantité, peut être recalculé : ce n'est donc pas une caractéristique de l'**ARTICLE**.

# Identifier de manière unique chaque entité

- Un **CLIENT** sera identifié par un Numéro unique, cette caractéristique de l'entité étant appelé `id_Client`.
- Une **FACTURE** sera identifiée par son Numéro qui est unique. Cette caractéristique sera appelée `id_Facture`.
- Un **ARTICLE** sera identifié par son Numéro qui est lui aussi unique. Cette caractéristique sera appelée `id_Article`.

# Etablir les relations et identifier les cardinalités

- Un **CLIENT** obtient une **FACTURE** qui contient des **ARTICLES** en certaine quantité.
- Un même **CLIENT** obtient 1 ou plusieurs **FACTURE**.
- Une même **FACTURE** est obtenue par un seul **CLIENT**.
- Une même **FACTURE** contient 1 ou plusieurs **ARTICLE**.
- Un **ARTICLE** est contenu dans 0 ou n **FACTURE**.



# Le modèle logique de données relationnelles

Relation (X,1)-(X,n) entre FACTURE et CLIENT :

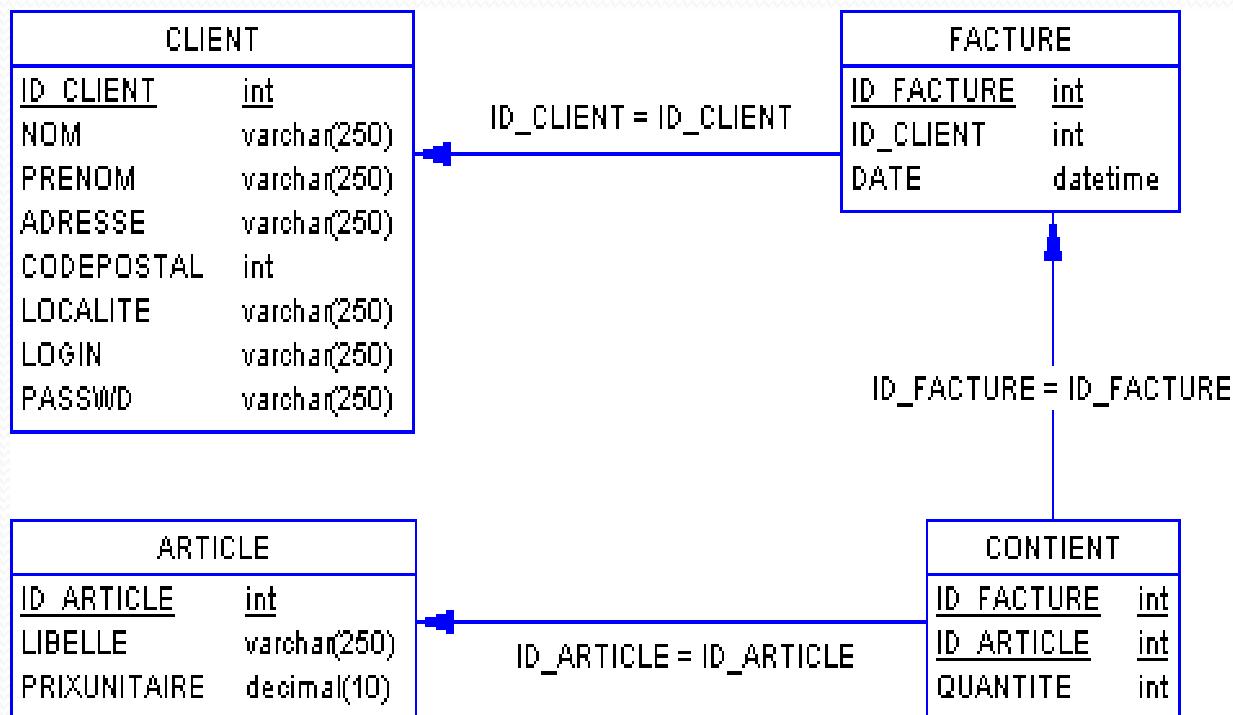
CLIENT (id\_Client, Nom, Prenom, Adresse, CodePostal, Localite, Login, Passwd)  
FACTURE(id\_Facture, #id\_Client, Date)

Relation (X,n)-(X,n) entre FACTURE et ARTICLE :

CONTIENT(#id\_Facture, #id\_Article, Quantite)  
ARTICLE(id\_Article, Libelle, PrixUnitaire)

| (id\_Newsletters, #id\_Rubrique, Sujet, DateEnvoie, Contenu)

# Le modèle physique de données



# Comparatif UML et Merise

# UML

L'UML (Unified Modeling Language : langage de modélisation objet uniifié) est une plus qu'une méthode de modélisation, c'est un langage à part entière dédié à l'objet.

L'approche est faites par des diagrammes destinés à modéliser plusieurs domaines :

- Les diagrammes structurels
- Les diagrammes comportementaux
- Les diagrammes d'interaction

# UML

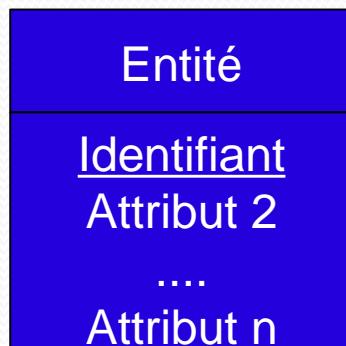
Parmi tous les diagrammes proposés par la méthode UML, seul le diagramme de classes permet de modéliser les bases de données.

Les autres diagrammes ne seront donc pas expliqués dans ce cours.

# Entité, Classe

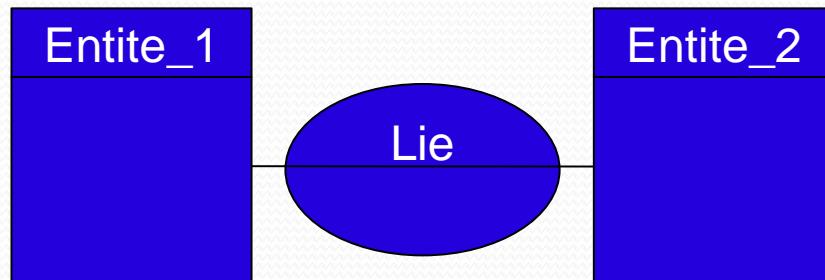
Merise

UML

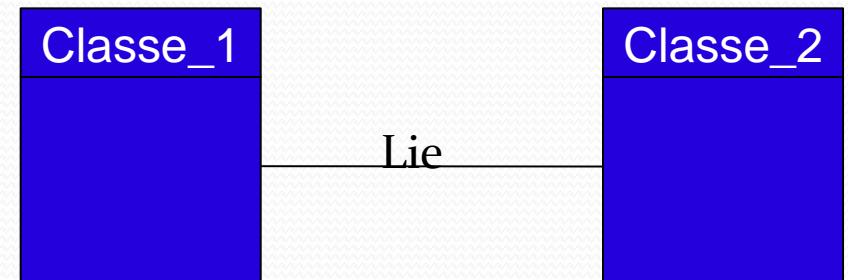


# L'association

Merise



UML



# Cardinalités, Multiplicités

Merise

UML

Cardinalités	Lecture
0 , 1	Lien vers 0 ou 1
1 , 1	Lien vers 1
0 , n	Lien vers 0 ou N
1 , n	Lien vers 1 ou N

Multiplicités	Lecture
0..1	Lien vers 0 ou 1
1	Lien vers 1
*	Lien vers 0 ou plusieurs
1..*	Lien vers 1 ou plusieurs

# Les associations un à un

Merise

UML

Cardinalités		Multiplicités	
0 , 1	0 , 1	0 .. 1	0 .. 1
0 , 1	1 , 1	0 .. 1	1
1 , 1	1 , 1	1	1

# Les associations un à plusieurs

Merise

UML

Cardinalités		Multiplicités	
0 , 1	0 , N	0 .. 1	*
0 , 1	1 , N	0 .. 1	1 .. *
1 , 1	0 , N	1	*
1 , 1	1 , N	1	1 .. *

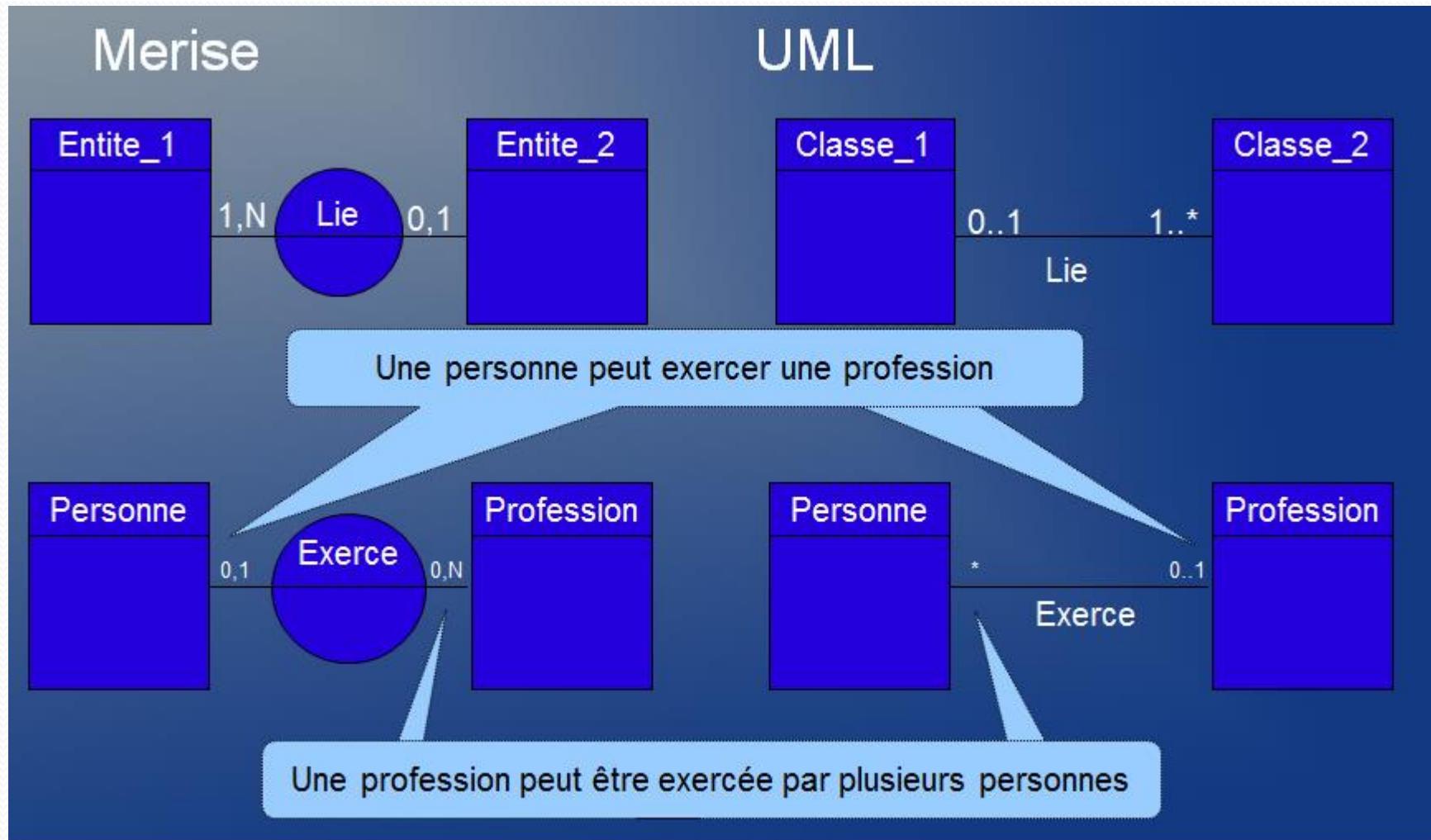
# Associations plusieurs à plusieurs

Merise

UML

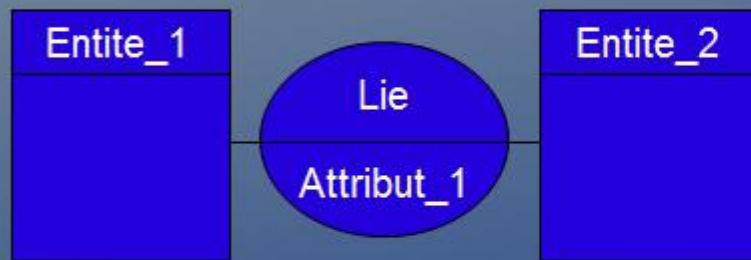
Cardinalités		Multiplicités	
0 , N	0 , N	*	*
0 , N	1 , N	*	1 .. *
1 , N	0 , N	1 .. *	*
1 , N	1 , N	1 .. *	1 .. *

# Exemple d'association

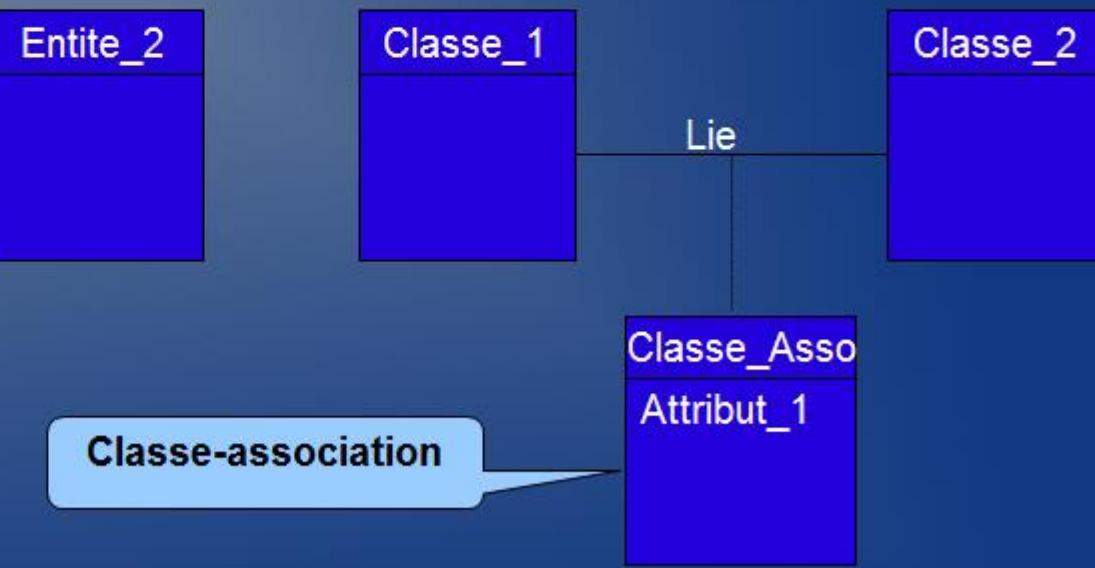


# Association avec attributs

Merise



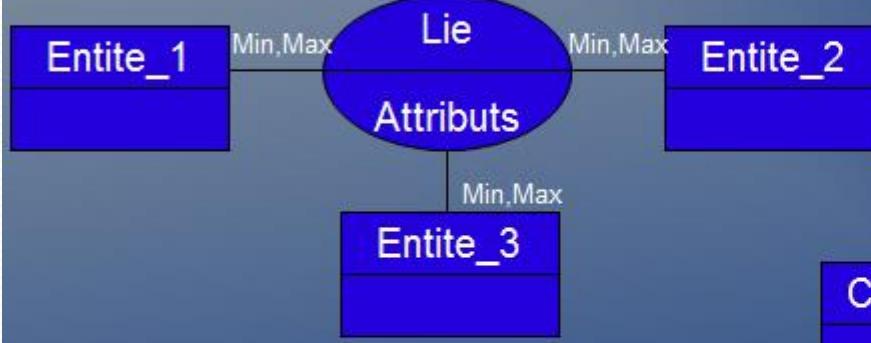
UML



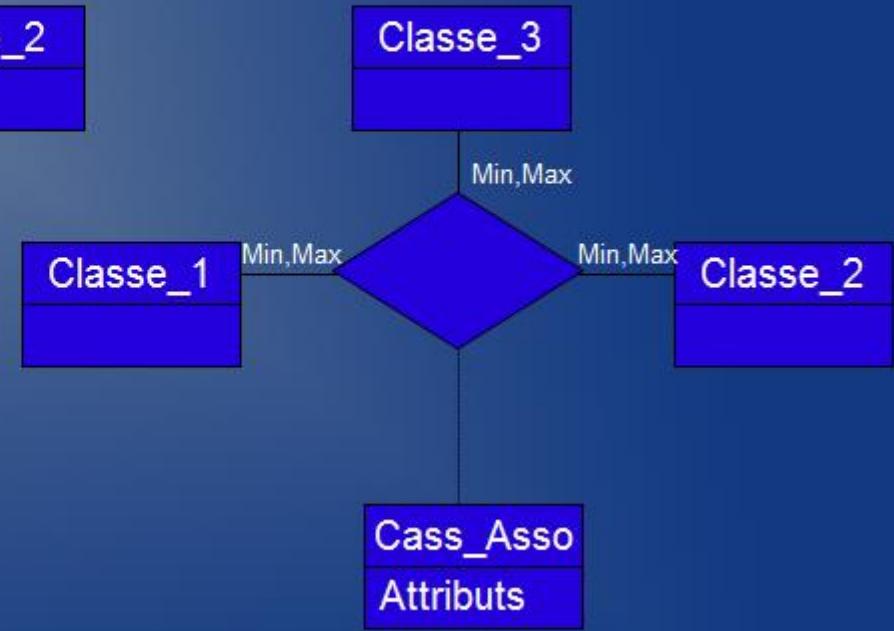
Classe-association

# L'association ternaire

Merise

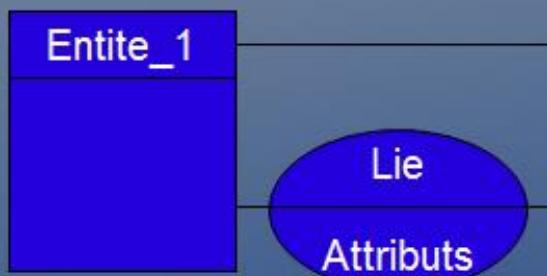


UML

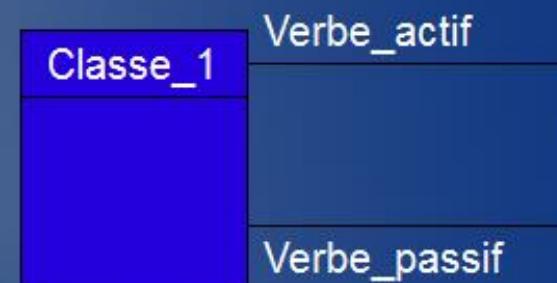


# L'association réflexive

Merise



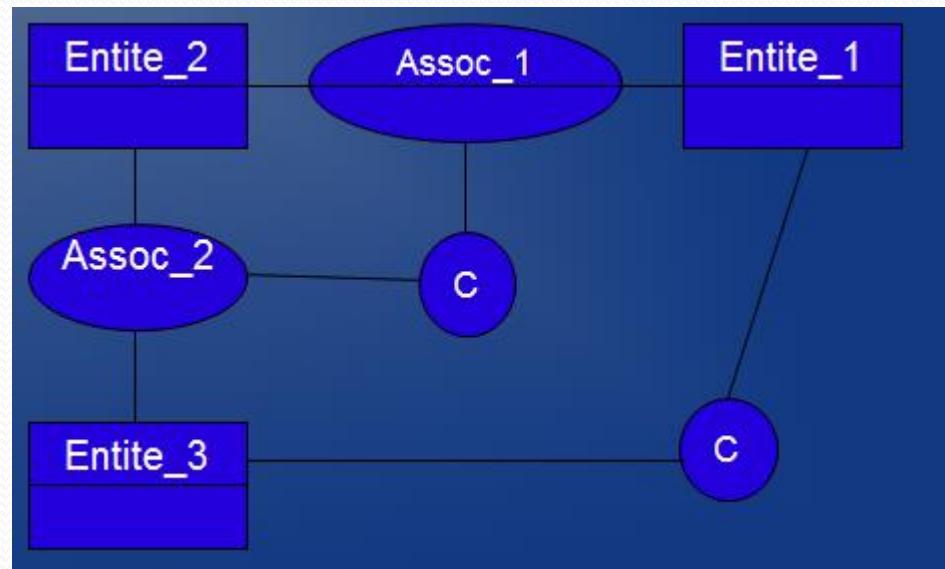
UML



# Les contraintes

Afin de prendre en compte plus de règles, la norme Merise 2 intègre les notions de contraintes.

Les contraintes associent des entités ou des relations afin de préciser des règles supplémentaires.



# Les contraintes

Les contraintes les plus courantes sont :

- L'exclusivité : peut être l'un ou l'autre
- La totalité : doit être l'un, l'autre ou les deux
- La partition : doit être l'un ou l'autre
- L'inclusion : ce qui est dans l'un est dans l'autre
- La simultanéité : doit être dans l'un et dans l'autre
- L'unicité : ce qui est dans l'un ne peut être qu'une fois dans l'autre

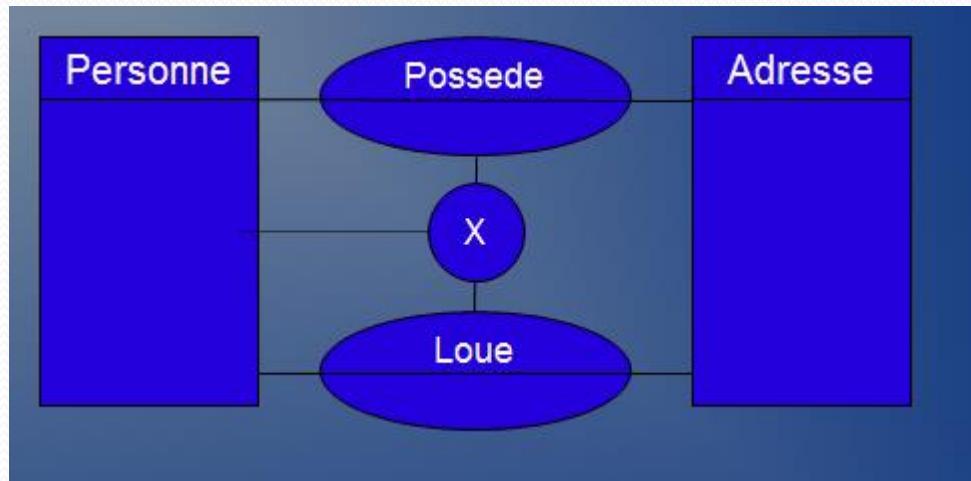
# Les contraintes

Les contraintes sont modélisés par des codes :

- L'exclusivité : X
- La totalité : T
- La partition : XT (parfois P)
- L'inclusion : I
- Simultanéité : S
- L'unicité : U

# Les contraintes

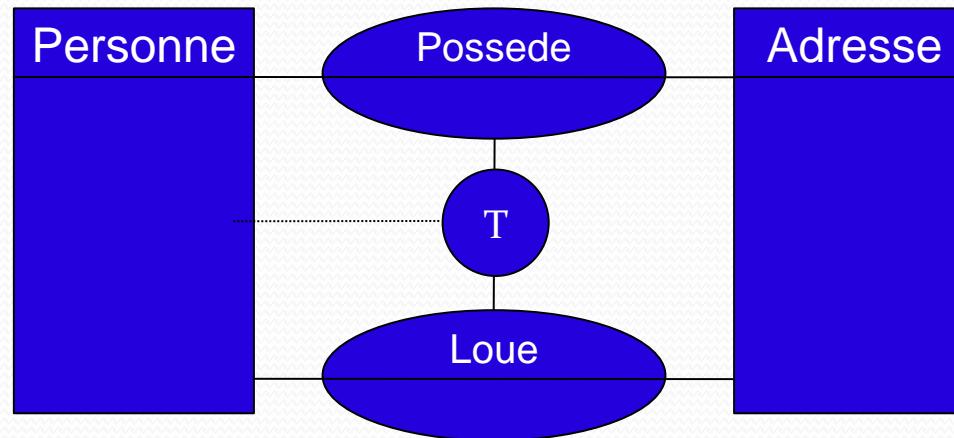
Exemple de contrainte d'exclusivité :



Ici, on modélise le fait qu'une personne ne peut pas posséder et louer une même adresse.

# Les contraintes

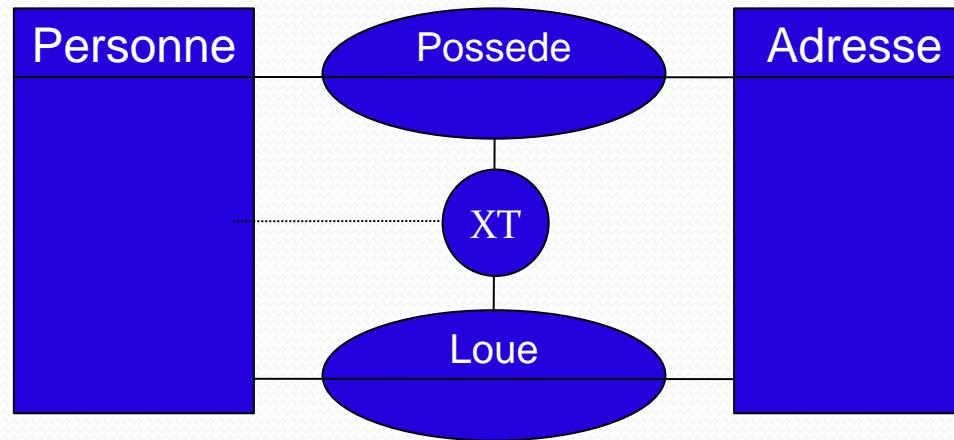
Exemple de contrainte de totalité :



Ici, on modélise le fait qu'une personne est forcément propriétaire, locataire ou les deux.

# Les contraintes

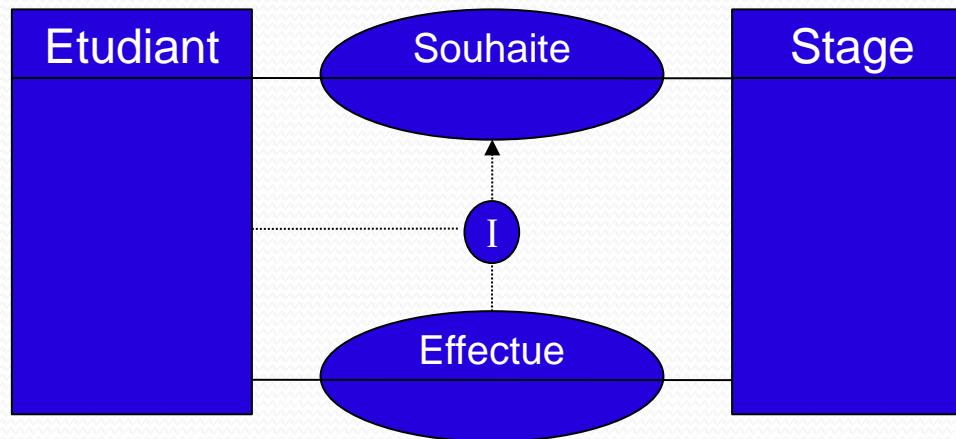
Exemple de contrainte de partition :



Ici, on modélise le fait qu'une personne est forcément propriétaire ou locataire mais pas les deux.

# Les contraintes

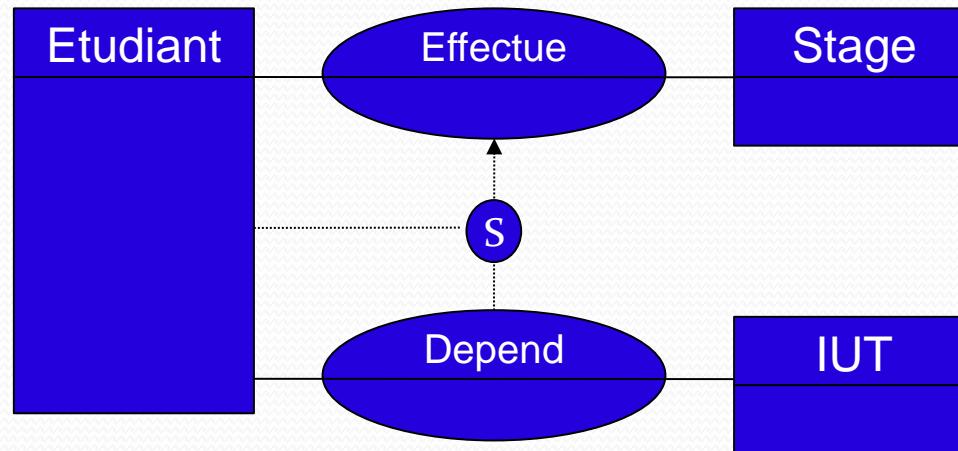
Exemple de contrainte d'inclusion :



Ici, on modélise le fait que le stage qu'un étudiant effectuera est un des stage qu'il a souhaité.

# Les contraintes

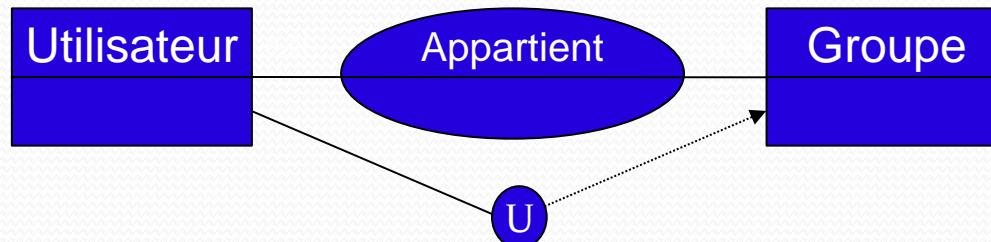
Exemple de contrainte de simultanéité :



Ici, on modélise le fait que le stage est obligatoire pour tous les étudiants puisque toute occurrence dans 'Depend' doit exister dans 'Effectue'.

# Les contraintes

Exemple de contrainte d'unicité :



Ici, on modélise le fait qu'un utilisateur ne peut pas appartenir plusieurs fois au même groupe.

Cette contrainte n'a d'intérêt que dans une relation plusieurs à plusieurs.

# L'agrégation

L'agrégation est une notion proposée par le langage UML.

Elle représente des associations qui ne sont pas symétriques et pour lesquelles une classe joue un rôle prépondérant par rapport à l'autre.

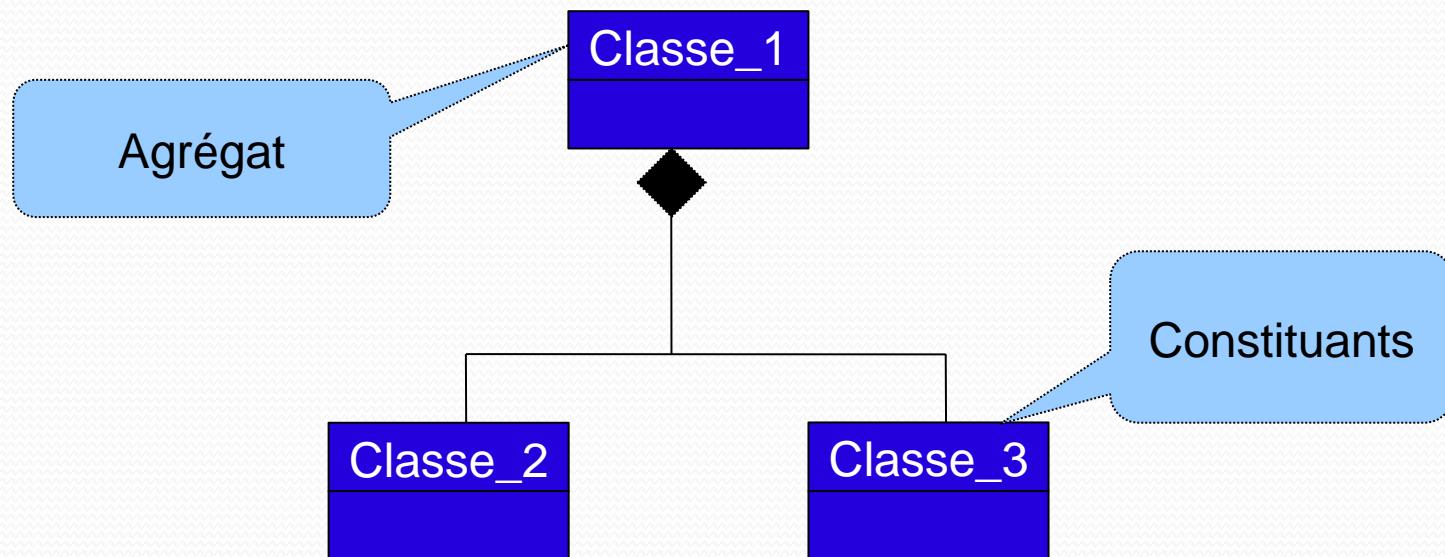
Il existe deux formes d'agrégation :

- La composition
- L'agrégation partagée

# La composition

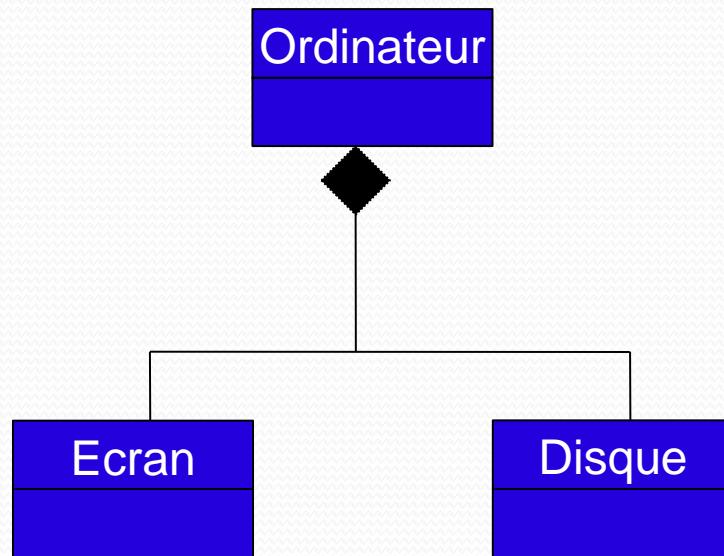
La composition est utilisée pour préciser qu'une classe est un sous-ensemble d'une autre.

Elle se caractérise par un losange plein.



# La composition

Exemple :

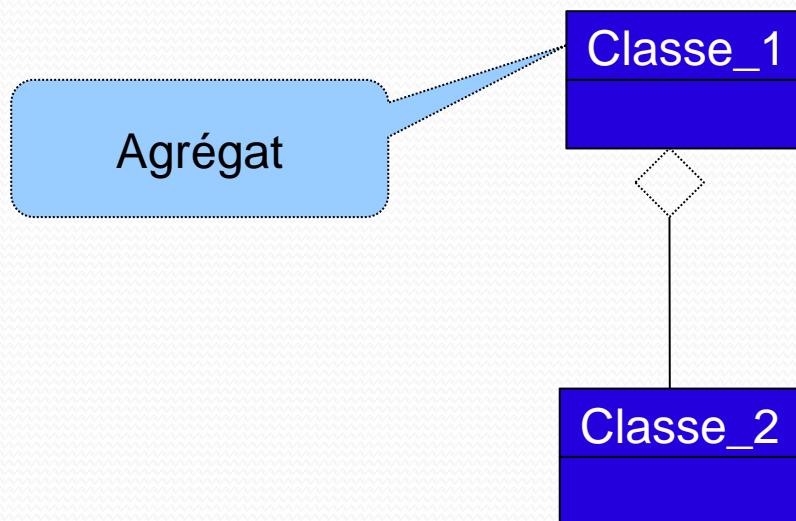


La composition implique que lors de la suppression d'un ordinateur, l'écran et les disques associés seront supprimés.

# L'agrégation partagée

L'agrégation partagée permet de préciser qu'une classe est dépendante d'une autre.

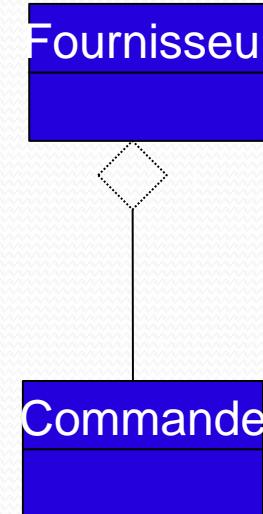
L'agrégation partagée se modélise par un losange clair.



# L'agrégation partagée

Exemple :

Une commande n'a de sens  
que si elle est liée à  
un fournisseur.



En cas de suppression du fournisseur, la commande  
n'est pas nécessairement supprimée.

# L'héritage

L'héritage existe en Merise 2 et en UML (1 & 2).

Le concept est le même qu'en langage objet.

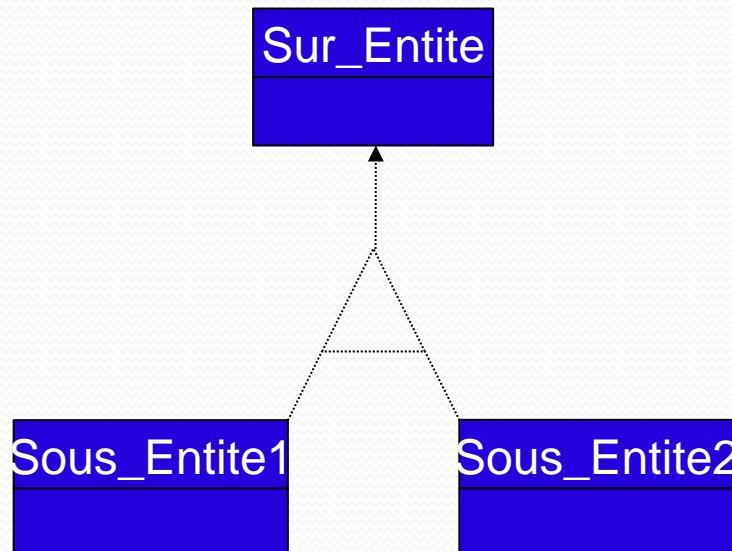
L'héritage permet de mutualiser des attributs tout en prenant en compte les spécificités des héritiers.

On peut définir le besoin d'héritage en se demandant si l'héritier est composé du parent.

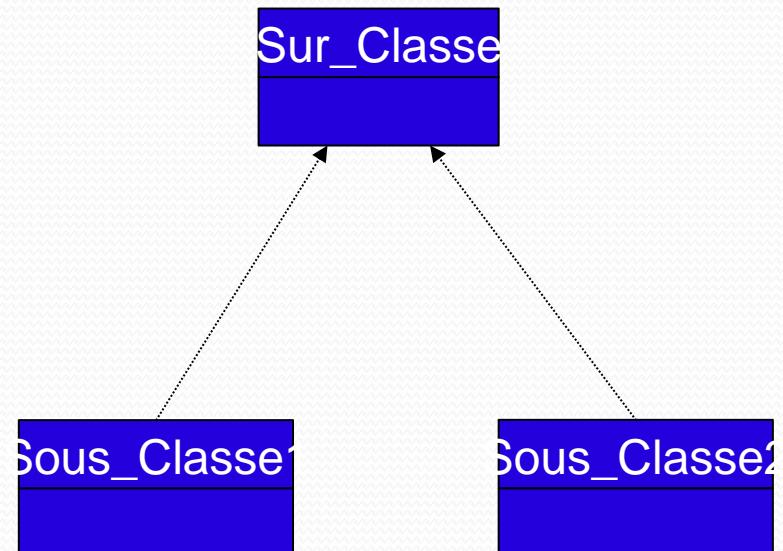
On modélise l'héritage en Merise par un triangle contenant les contraintes, en UML par une flèche de l'héritier vers le parent.

# L'héritage

Merise

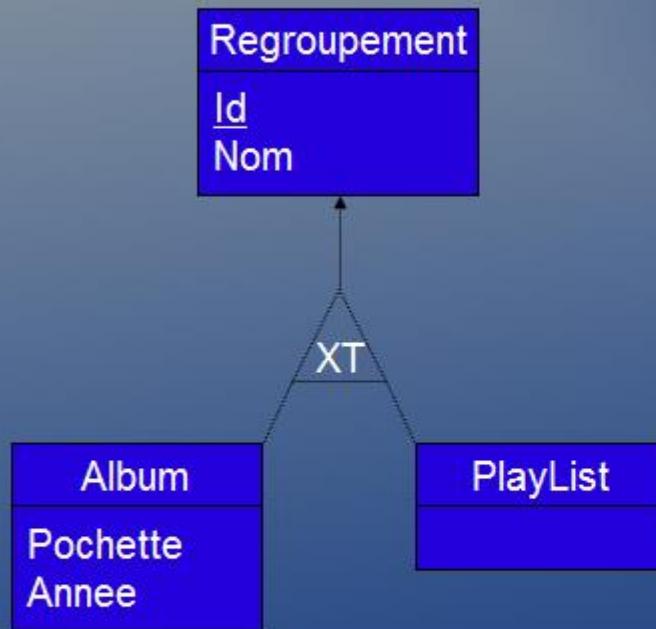


UML

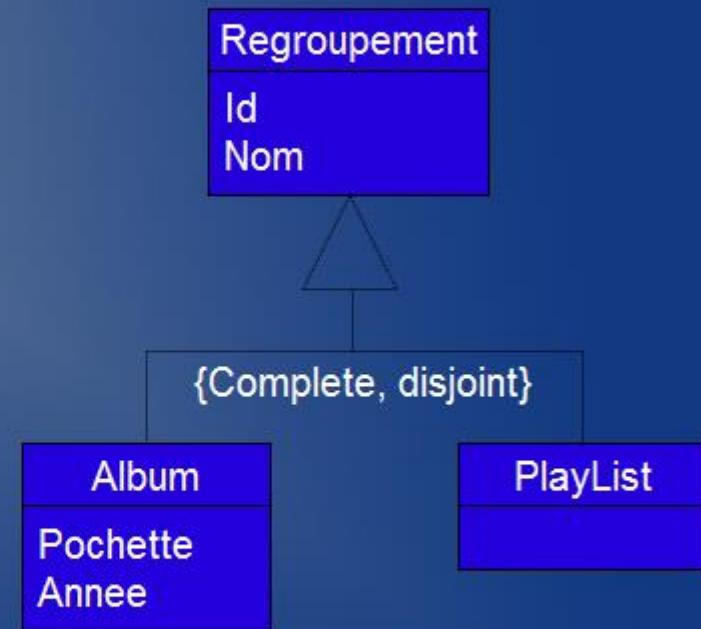


# L'héritage

Merise



UML



Exemple d'héritage de partition :

Regroupement est un album ou une playlist.

# Passage au niveau logique

Le niveau logique est la représentation des tables telles qu'elles seront dans la base de données.

Le niveau logique n'est pas impacté par le système de bases de données utilisé.

Nous présenterons les structures ainsi :

Table[cle\_primaire,champ\_N, cle\_etrangere#]

# Passage au niveau logique

Nous savons déjà que :

- Une entité ou une classe devient une table.
- L'identifiant Merise devient la clé primaire, pour UML il faut la définir.
- Une association 1 à 1 ou 1 à plusieurs se traduit par une clé étrangère dans la table ayant la cardinalité ou la multiplicité maximum à 1.
- Une association plusieurs à plusieurs se traduit par une nouvelle table.

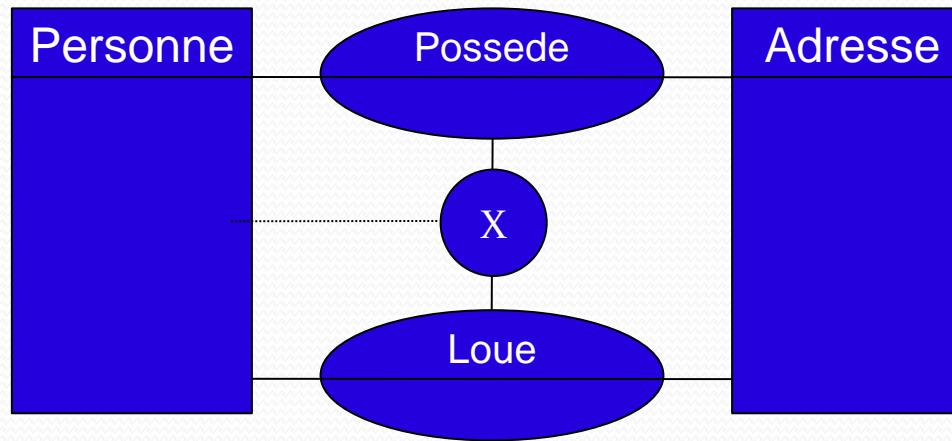
# Passage au niveau logique (contraintes)

Exemple de prise en compte :

Contrainte	Logique	Physique
Exclusivité (X)	Fusion des tables avec champ indicateur	Le champ indicateur NULL
Totalité (T)	Fusion des tables avec champ indicateur	Le champ indicateur NOT NULL
Partition (XT)	Fusion des tables avec champ booléen	Le champ booléen NOT NULL
Inclusion (I)	Fusion des tables avec champs NULL	Les champs inclus sont NULL
Simultanéité (S)	Fusion des 2 tables	Les champs sont NOT NULL
Unicité (U)		Indexe unique

# Passage au niveau logique (contraintes)

Exemple avec la contrainte d'exclusivité :



Personne [idp, nom, prenom, date\_naiss]

Adresse [ida, voie, cp, ville]

Dispose [idp#, ida#, ind\_loue]

# Passage au niveau logique (contraintes)

Exemple avec la contrainte d'exclusivité :

Personne [idp, nom, prenom, date\_naiss]

Adresse [ida, voie, cp, ville]

Dispose [idp#, ida#, ind\_loue]

Personne	
<u>Idp</u>	<u>BIGINT</u>
Nom	vachar(50)
Prenom	vachar(50)
date_naiss	date

Dispose	
<u>Idp#</u>	<u>BIGINT</u>
<u>Ida#</u>	<u>BIGINT</u>
Ind_loue	char(1) NULL

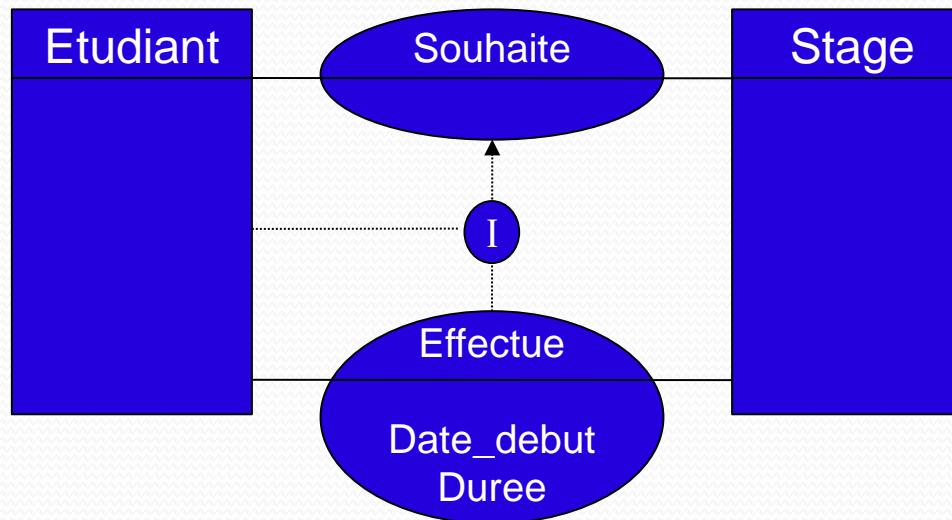
Adresse	
<u>Ida</u>	<u>BIGINT</u>
Voie	vachar(100)
CP	char(5)
Ville	varhcar(50)

Le champ 'Ind\_loue' pourra prendre les valeurs :

'L' : locataire, 'P' : propriétaire, NULL : pas précisé

# Passage au niveau logique (contraintes)

Exemple avec la contrainte d'inclusion :



Etudiant [ide, nom, prenom, date\_naiss]

Stage [ids, nom]

Desire [ide#, ids#, effectue, date\_deb, duree]

# Passage au niveau logique (contraintes)

Exemple avec la contrainte d'inclusion :

Etudiant [ide, nom, prenom, date\_naiss]

Stage [ids, nom]

Desire [ide#, ids#, effectue, date\_deb, duree]

Etudiant	
ide	BIGINT
Nom	vachar(50)
Prenom	vachar(50)
date_naiss	date

Desire	
ide#	BIGINT
ids#	BIGINT
Effectue	TINYINT NULL
date_deb	DATE NULL
DUREE	INT NULL

Stage	
ids	BIGINT
nom	vachar(50)

Pour prendre en compte le fait qu'on ne peut effectuer qu'un seul stage, il faudra créer un indexe unique sur les champs 'ide', 'ids', 'effectue'.

# Passage au niveau logique (agrégation)

L'agrégation va se traduire par une contrainte d'intégrité référentielle au niveau de la base de données .

La composition va utiliser 'ON DELETE CASCADE' puisque la suppression d'un agrégat entraîne la suppression des constituants.

L'agrégation partagée va s'assurer que la valeur référencée existe bien par le biais de la déclaration de la clé étrangère et d'une action de type cascade, set null.

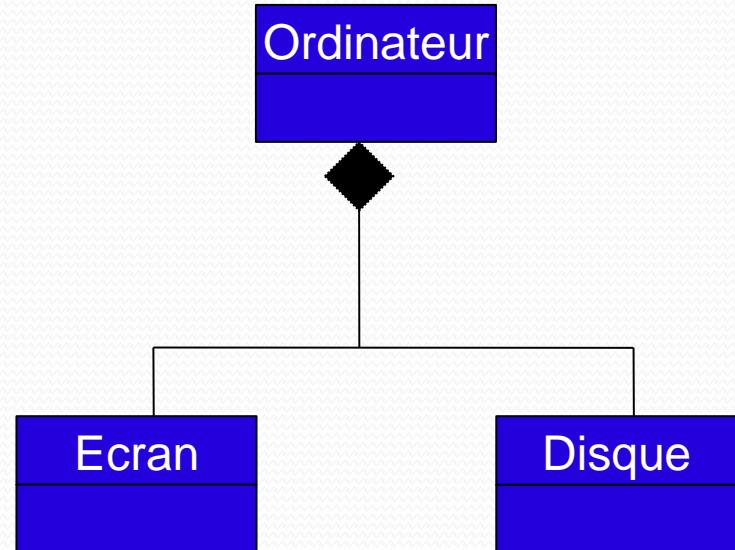
# Passage au niveau logique (agrégation)

Exemple :

Ordinateur[ido, nom]

Ecran[ide, nom, type, taille]

Disque[idd, nom, capacite]



Lors de la création des tables, on ajoutera pour 'Ecran' et 'Disque' :

foreign key (ido) references ordinateur(ido)  
on delete cascade

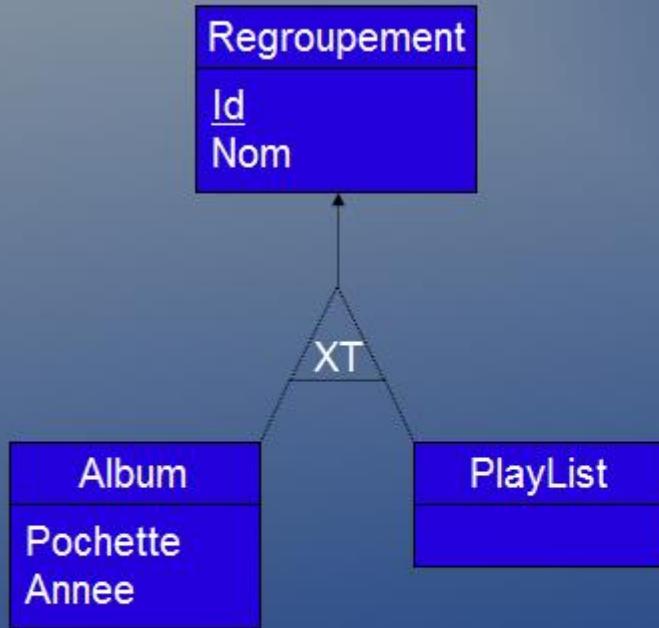
# Passage au niveau logique (héritage)

Il existe trois méthodes de décomposition pour traduire une association d'héritage :

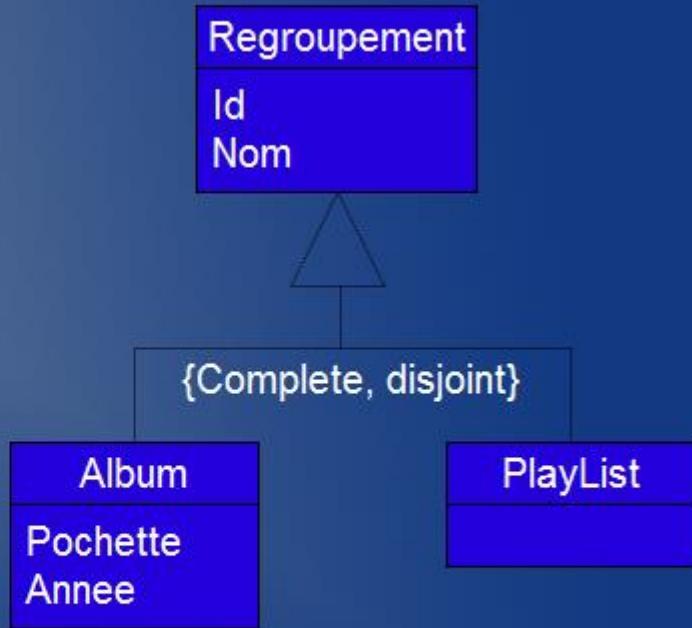
- La décomposition par distinction (une table par entité ou classe)
- La décomposition descendante (une table par sous-entité ou sous-classe)
- La décomposition ascendante (une table par sur-entité ou sur-classe)

# Passage au niveau logique (héritage)

Merise



UML



Décomposition ascendante :

Regroupement[id, nom, prochette, annee]

# Passage au niveau logique (héritage)

Décomposition par distinction :

Regroupement [idr, nom]

Album [idr#, pochette, annee]

PlayList [idr#]

Décomposition descendante :

Album [id, nom, pochette, annee]

PlayList [id, nom]