



Conception de bases de données

Françoise Dubisy

Informatique de gestion

UE 221 : Analyse métier et conception de bases de données

Hénallux – Catégorie économique – 2019-2020

1	Qu'est-ce qu'une base de données ?	7
1.1	Historique	7
1.1.1	Fichiers séquentiels	7
1.1.2	Evolution des supports	7
1.1.3	Evolution des systèmes d'exploitation	8
A.	Mémoires et périphériques d'entrée-sortie	8
B.	Multi-tâches	8
1.1.4	Evolution des méthodes d'accès aux données et de leur organisation	8
1.1.5	Systèmes documentaires	9
1.1.6	Systèmes de gestion de bases de données	9
1.2	Inconvénients des systèmes de fichiers	10
1.3	Liens entre les données	12
1.4	Caractéristiques d'une base de données (B.D.)	18
1.5	Difficultés d'identification des concepts et des liens	19
1.6	Systèmes de gestion de bases de données (S.G.B.D.)	23
1.6.1	Le programmeur	23
1.6.2	L'utilisateur	24
1.6.3	L'administrateur	24
1.6.4	Le S.G.B.D.	25
2	Le schéma conceptuel	28
2.1	Introduction	28
2.2	Entités	30
2.3	Associations	32
2.3.1	Qu'est-ce qu'une association ?	32

2.3.2	Cardinalités maximales	36
A.	Relation de type 1 à 1	37
B.	Relation de type 1 à N.....	38
C.	Relation de type N à N	39
2.3.3	Cardinalités minimales	39
2.3.4	Exemples	41
2.4	Construction d'un schéma conceptuel.....	44
2.4.1	Décomposition de l'énoncé.....	44
2.4.2	Représentation d'une proposition	46
A.	Concepts	46
B.	Liens.....	47
2.5	Intérêt de prévoir une association plutôt qu'un attribut.....	49
2.6	Attributs	51
2.6.1	Identifiant ("primary key").....	51
2.6.2	Obligatoire ou facultatif	51
2.6.3	Atomique ou décomposé.....	53
2.6.4	Attribut facultatif versus attribut booléen	54
2.7	Non redondance dans le schéma entités-associations	58
2.8	Pertinence des propositions	62
2.8.1	Contradiction des propositions.....	62
2.8.2	"Bruit" dans les propositions	62
2.8.3	Autres suggestions	63
2.9	Associations particulières	65
2.9.1	Relation de degré supérieur à 2	65

2.9.2	Association cyclique	71
2.9.3	Relation avec attributs	74
2.10	Contraintes additionnelles	78
2.10.1	Identifiant composé d'attributs	78
2.10.2	Identifiant hybride	81
2.10.3	Identifiant d'association implicite.....	86
2.10.4	Autres contraintes additionnelles.....	87
2.11	Dépendances fonctionnelles	88
2.12	Transformation de schéma	94
2.12.1	Transformation d'une association en une entité	94
2.12.2	Transformation d'attribut en entité	98
3	Bases de données relationnelles	102
3.1	Structure de base : les tables	102
3.1.1	Table.....	102
3.1.2	Ligne.....	103
3.1.3	Colonne	103
3.1.4	Valeur	104
3.2	Notion d'ordre	105
3.2.1	Ordre des lignes.....	105
3.2.2	Ordre des colonnes.....	105
3.3	Identifiant ou clé primaire	107
3.4	Traduction d'un schéma conceptuel en relationnel.....	109
3.4.1	Représentation des entités et des attributs	109
3.4.2	Représentation des associations.....	110

A. Représentation des associations 1 à N.....	110
B. Représentation des associations N à N	115
C. Représentation des associations cycliques	119
D. Représentation des relations de degré supérieur à 2	123
3.5 Conclusion	125
4 Héritage et bases de données relationnelles	126

Partie 1

Introduction aux bases de données

1 Qu'est-ce qu'une base de données ?

Un ordinateur est une machine à traiter des informations ; d'où l'importance des données. Les données doivent être organisées et stockées en vue de leur manipulation et interrogation ultérieures.

1.1 Historique

1.1.1 Fichiers séquentiels

A l'origine, les données sont stockées sur des cartes perforées. Ensuite, apparaissent les bandes magnétiques. Ces deux types de supports ne permettent l'exploitation des données que de façon séquentielle.

De plus, un fichier ne reprend que les données *relatives à **une seule** application*.

Si plusieurs applications utilisent les mêmes informations, celles-ci se voient dupliquées entièrement ou partiellement dans plusieurs fichiers, ce qui entraîne une **redondance** d'information.

Le principal inconvénient de cette duplication d'information est le **risque d'incohérence**. Une donnée peut être modifiée dans un fichier, sans que la modification soit répercutée correctement dans tous les fichiers contenant l'information. Cela vient du fait qu'**un fichier est dépendant d'une application** ; c'est donc via cette application que toute modification d'information sera répercutée dans le fichier.

En outre, il est **impossible d'interroger** ces données **dispersées** dans différents fichiers **sans aucun lien entre eux**.

1.1.2 Evolution des supports

L'organisation des fichiers évolue : le **temps d'accès** aux données **diminue** (pointeurs, index...).

Mais chaque application gère toujours ses propres fichiers, d'où toujours le risque de *redondance* et d'*incohérence* des données.

1.1.3 Evolution des systèmes d'exploitation

Les systèmes d'exploitation ont ensuite mieux géré les ressources physiques.

A. Mémoires et périphériques d'entrée-sortie

Le programmeur est de plus en plus déchargé des tâches de gestion du matériel périphérique.

Exemples : gestion des imprimantes, gestion des disques, adresse physique des fichiers...

B. Multi-tâches

Le **partage de l'unité centrale** (processeur) par **plusieurs utilisateurs** ("time-sharing") permet une meilleure utilisation du processeur.

Non seulement le processeur peut être partagé, mais également **un même programme**. Dans ce cas, une seule copie du code est présente en mémoire centrale, mais elle est exécutée par plusieurs utilisateurs.

En parallèle avec les systèmes d'exploitation, ont évolué les méthodes d'accès aux données et leur organisation.

1.1.4 Evolution des méthodes d'accès aux données et de leur organisation

Un même fichier de données peut être partagé par plusieurs utilisateurs. Un gestionnaire de fichier est alors indispensable pour éviter tout conflit d'accès.

Apparaissent alors les **systèmes transactionnels** reposant sur la notion de transaction. Une transaction est une suite logique d'instructions considérée comme formant un tout.

Il y a différentes façons de gérer l'accès à un fichier par un utilisateur :

- ① soit bloquer tout le **fichier** ;
- ② soit bloquer **un seul enregistrement à la fois** ;
- ③ soit adopter le **principe de la transaction** :

Scénario de mise à jour d'un enregistrement

Début transaction

lire l'enregistrement → enregistrement bloqué uniquement le temps d'en faire une copie (copie1)

modifier l'enregistrement → création d'une seconde copie (copie2) ;
puis application des modifications à la copie2

Fin transaction

→ pour s'assurer que l'enregistrement du fichier n'a pas été modifié entretemps, une troisième copie (copie3) est faite :
si copie1 = copie3 alors écriture dans le fichier de copie2
sinon avertir l'utilisateur

1.1.5 Systèmes documentaires

Il s'agit d'un type de gestionnaire de fichiers plus élaboré.

En 1969, apparaît le concept de **banque de données**. Il s'agit de **données mises à la disposition d'un grand nombre d'utilisateurs**. Ces données sont cependant **faiblement structurées**. La consultation se fait uniquement via des **mots-clés** et des **dictionnaires**.

Exemple : répertoires de références bibliographiques

1.1.6 Systèmes de gestion de bases de données

En 1970, IBM propose un système *qui dispense le programmeur de naviguer jusqu'à l'information désirée*. C'est le **système de gestion de base de données** qui se charge de trouver le *chemin d'accès optimal*.

Exemple : systèmes relationnels (langage SQL)

1.2 Inconvénients des systèmes de fichiers

① Dépendance des applications

Chaque application possède ses propres fichiers. Chaque fichier est généralement mis à jour par l'application qui l'a créé.

② Redondance des données

Même si deux applications ont besoin des **mêmes informations**, celles-ci se trouveront **dans deux fichiers** différents souvent **de structures différentes**.

③ Risque d'incohérence des données

La **duplication** des données dans des fichiers différents entraîne un risque d'**incohérence** si la modification d'une donnée n'est pas répercutée dans tous les fichiers.

④ Impossibilité d'interroger les données

Les données sont dispersées dans différents fichiers qui n'ont aucun lien entre eux.

⑤ Sécurité

Chaque utilisateur **sauve ses propres fichiers** avec une fréquence qui lui est propre.

⑥ Pas de modification de structure possible

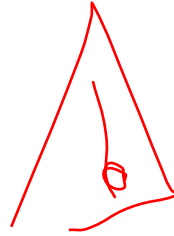
Une fois un fichier créé, il est **impossible de modifier la définition des types d'enregistrement**.

Exemple : ajouter/retirer un champ à tous les enregistrements d'un fichier.

Si une telle modification doit cependant avoir lieu, le seul recours pour le programmeur est de modifier le programme et de copier le fichier dans un nouveau fichier ayant la structure voulue.

Mais l'inconvénient principal est le suivant.

⑦ **Pas de relations entre fichiers** 💣

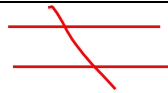


Il est difficile de mettre en relation les données de deux fichiers différents. On ne peut pas établir de relations logiques entre fichiers en vue de les exploiter via des interrogations (cf section 1.3).

1.3 Liens entre les données

Quelques définitions :

DONNEE : **Enregistrement** dans un **code** donné d'un objet, un texte, un concept, un fait... en vue de transmettre ou stocker de l'information, interpréter ou effectuer un traitement par l'homme ou la machine, ou encore en déduire de nouvelles informations



INFORMATION : **Sens, signification** que l'on attache à ou que l'on déduit d'un ensemble de données

L'aspect **subjectif** de la notion d'information est à souligner.

FICHER : **Ensemble de données** utilisables en général par une seule application

Supposons que l'on doive gérer une bibliothèque et qu'on nous demande de pouvoir répondre aux questions ci-dessous. Comment structurer les données ?

① Etant donné un ouvrage, trouver le nom de l'auteur

⇒ Prévoir un ensemble de **fiches** **Ouvrage**

Exemple :

Titre : *Le lotus bleu*
Catégorie : *BD*
Editeur : *Casterman*
Auteur : *Hergé*

Toujours minimiser le nbr
de de boîtes

Il suffit de prévoir la caractéristique **Auteur** sur la fiche de chaque ouvrage.

② Etant donné un ouvrage, retrouver toutes les caractéristiques de l'auteur

☹ Recopier toutes les caractéristiques de l'auteur sur les fiches de chacun de ses ouvrages !

Cela induit de la **redondance**. En effet, les informations concernant un même auteur seront **dupliquées**. Par exemple, les informations concernant *Hergé* se retrouveront enregistrées autant de fois dans la base de données qu'il y a d'ouvrages de *Hergé* dans la bibliothèque.

Cette redondance est difficile à gérer ; en effet, la moindre modification concernant un auteur devra être effectuée à plusieurs endroits dans la base de données. Le moindre oubli placera la base de données dans un état incohérent.

⇒ à éviter!

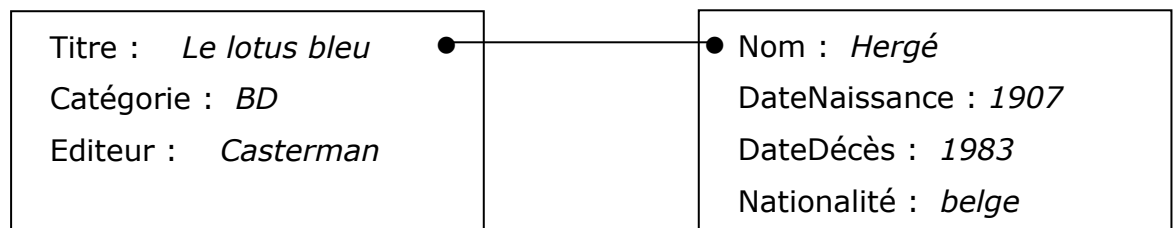


😊 Prévoir une fiche par auteur et relier l'ouvrage à son auteur

La recherche s'effectue en **deux étapes** :

- **rechercher** la **bonne fiche** ouvrage ;
- **retrouver** la **fiche de l'auteur correspondant**.

Exemple



On ne stocke qu'une fois les caractéristiques d'un auteur : il n'y a donc pas de problèmes de redondance. On relie ensuite la fiche auteur à une fiche ouvrage.

Il serait intéressant de disposer d'un système qui permette de gérer ce type de lien, à savoir :

A l'encodage :

A la création d'un nouvel ouvrage, il faudrait pouvoir préciser quel en est l'auteur, c'est-à-dire établir/mémoriser le lien entre la nouvelle fiche *Ouvrage* et la fiche *Auteur* correspondante.

De plus, il serait intéressant que dans un souci de cohérence, le système **refuse** de relier le nouvel ouvrage à un auteur **dont la fiche n'existe pas dans la base de données.**

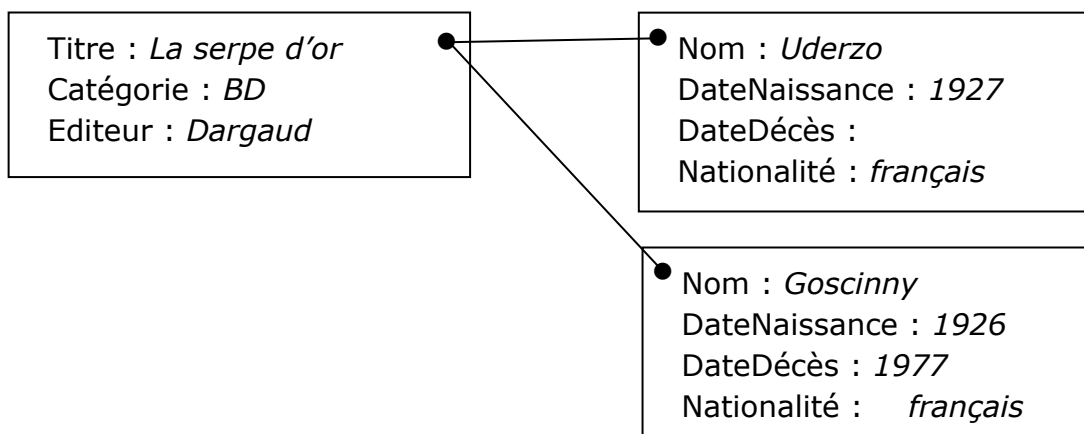
A l'utilisation (lors de la recherche d'information) :

Il faudrait un système capable, lors de l'interrogation de la base de données, d'exploiter ce lien, c'est-à-dire capable de retrouver toutes les caractéristiques de la fiche *auteur* liée à un *ouvrage* donné.

③ Etant donné un ouvrage, retrouver toutes les caractéristiques de ses auteurs

Le principe reste le même : on relie la fiche *Ouvrage* **aux fiches** correspondant aux auteurs de l'ouvrage.

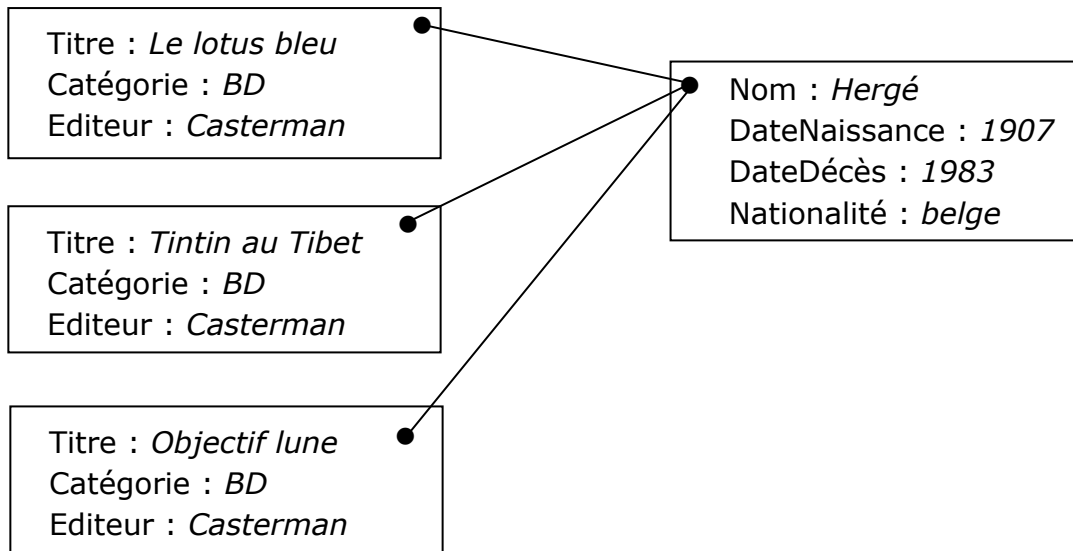
Exemples :



④ Etant donné un auteur, retrouver tous ses ouvrages

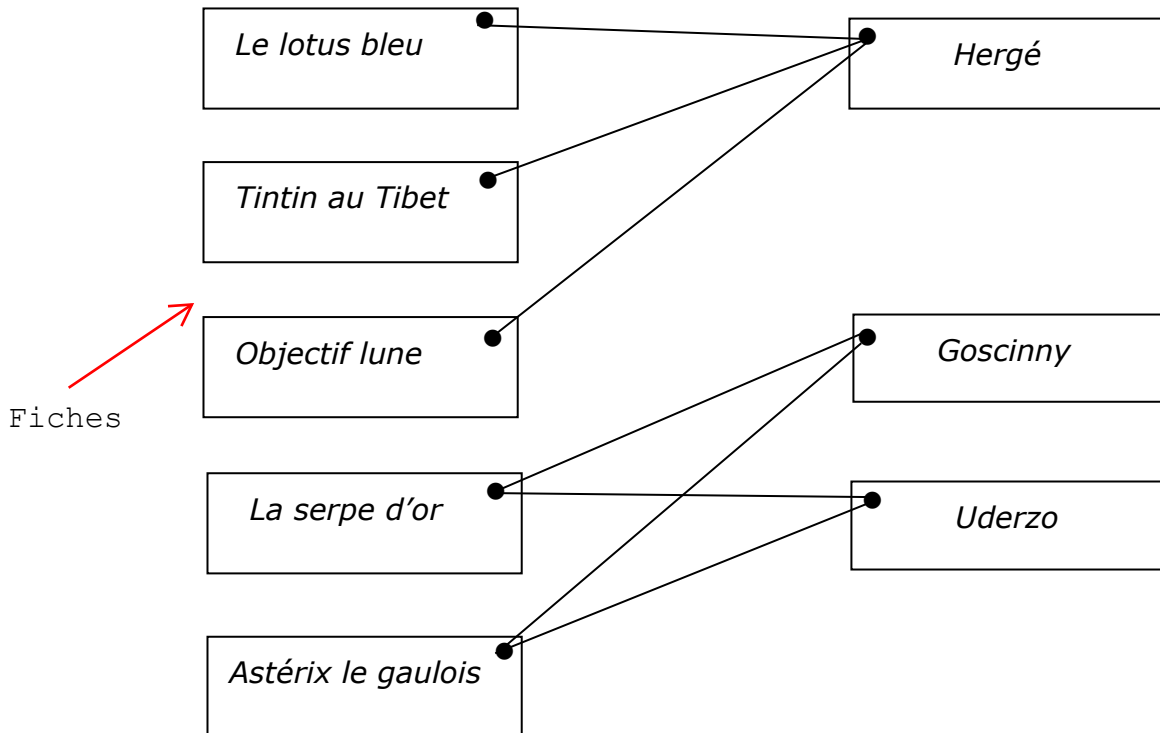
Il n'y a rien de plus à prévoir. Cette situation est une généralisation des points ① et ②.

Exemples :

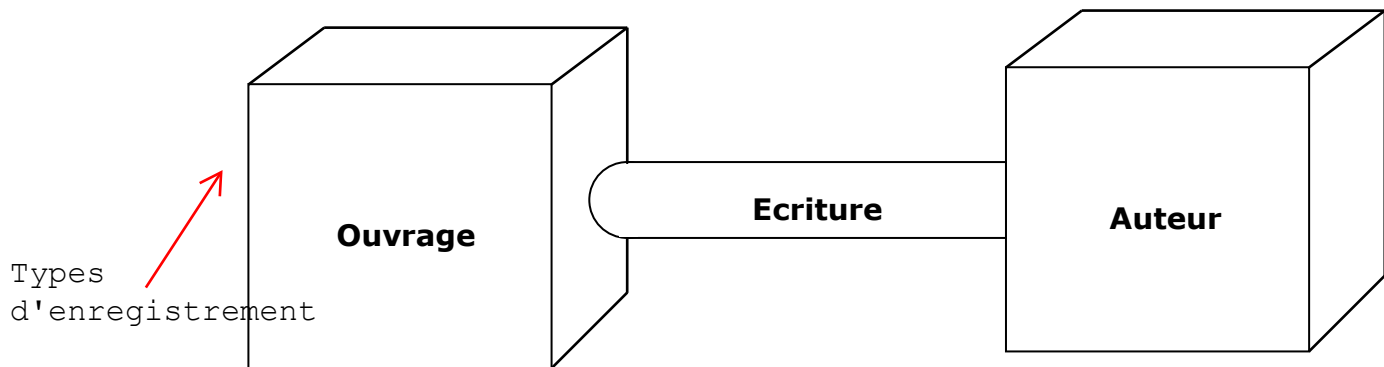


Par conséquent, une base de données est un ensemble de données reliées.

Au niveau fiches :




Au niveau types d'enregistrement :

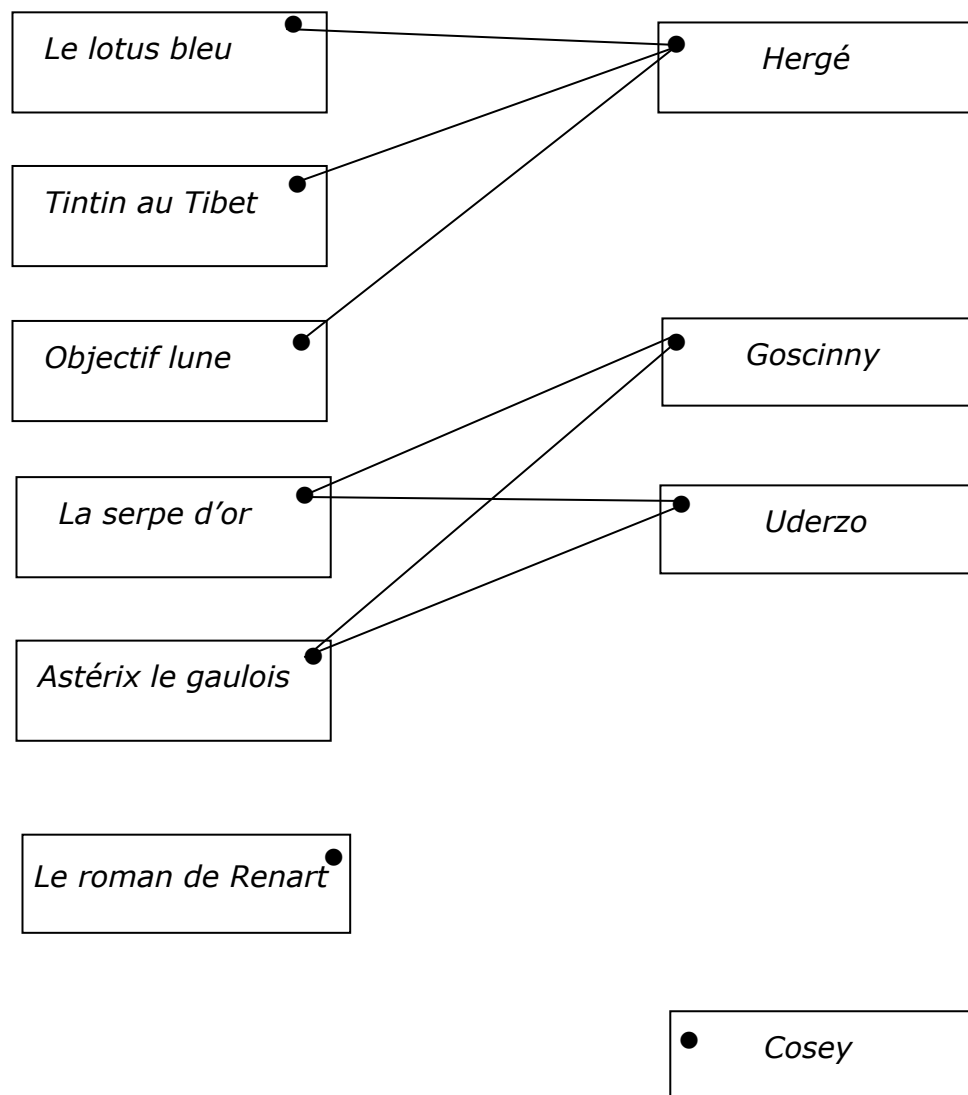


Lors de la création d'une base de données, on définit les **liens possibles** entre les objets.

Définir un lien *Ecriture* entre le type d'objets *Ouvrage* et le type d'objets *Auteur* c'est préciser qu'il est **possible** que des liens d'écriture existent entre des objets (fiches) qui sont du type *Ouvrage* et des objets (fiches) qui sont du type *Auteur*.

Cela ne signifie pas que tout objet du type *Ouvrage* (c'est-à-dire toute fiche ouvrage) a forcément au moins un lien avec un objet du type *Auteur* (c'est-à-dire une fiche auteur) et inversement. Par conséquent, pourrait être repris dans la base de données un ouvrage dont l'auteur est inconnu (ex : *le roman de Renart*) ou encore, pourrait être repris dans la base de données un auteur ainsi que ses coordonnées alors que la bibliothèque ne possède (encore) aucun de ses ouvrages (ex : *Cosey*). 

La base de données pourrait contenir alors l'ensemble des fiches suivant :



1.4 Caractéristiques d'une base de données (B.D.)

Quels sont les **objectifs d'une structuration des données sous forme de base de données** ? Une base de données aura les **caractéristiques** suivantes.

① **Relations** entre les données

Possibilité d'établir des relations entre les données (cf précédemment).

② **Sauvegarde** des données sur un support (disque)

Les données doivent pouvoir être **mémorisées** en vue d'une réutilisation future. Elles doivent également pouvoir être **facilement modifiables**.

③ **Partage** des données entre plusieurs utilisateurs

Les données sont **centralisées** et partageables (multi-users). Cela a pour conséquence une gestion de la **sécurité** et de la confidentialité des données via les mécanismes **des droits d'accès**.

④ **Indépendance** des données par rapport aux applications

Les données ne sont plus envisagées que pour une application déterminée.

⑤ **Sans redondance inutile**

Toute redondance d'information doit être évitée, **sauf** pour des raisons de **sécurité** et de **performance**.

⑥ **Contrôle de cohérence**

La nécessité d'un contrôle de cohérence est évidente **en cas de redondance** (une modification doit être répercutée automatiquement à plusieurs endroits).

De plus, des **contraintes** additionnelles peuvent accompagner la définition d'une base de données. Ces contraintes devront être respectées lors de toute modification de données de la B.D. C'est l'administrateur du système qui définit les contraintes additionnelles, mais c'est le système de gestion de la base de données qui se charge de les vérifier.

⑦ **Exploitation des données par interrogation**

Toutes les données d'une B.D. sont "directement accessibles" ; une B.D. est interrogeable sur **n'importe quel champ**. La recherche du **chemin d'accès optimal** n'est plus à la charge du programmeur.

⑧ **Longueur des enregistrements variable**

Dans un enregistrement, il peut y avoir des informations manquantes ou inconnues (valeur NULL).

N.B.

Ne pas confondre valeur **null** et "espaces" pour un champ alphanumérique.

Ne pas confondre valeur **null** et 0 pour un champ numérique.

⑨ **Modification possible de la structure des enregistrements**

Il est possible **d'ajouter, modifier** ou **supprimer un champ** d'un type d'enregistrement existant.

Une définition générale d'une base de données pourrait alors être :

Une B.D. est

un ensemble de données en relation, indépendantes des applications, sans redondance inutile, partageable entre plusieurs utilisateurs et dont on peut accéder à n'importe quel contenu en réponse à une question

1.5 Difficultés d'identification des concepts et des liens

= Résumé des pièges qui peuvent être rencontrés

Reprenons la gestion d'une bibliothèque. Envisageons un ou l'autre scénario typique.

① **Des emprunteurs empruntent des exemplaires d'ouvrages**

On ne considère que les emprunts en cours.

Il faut par exemple pouvoir gérer les cas suivants :

- *Jules Dupond a emprunté l'unique exemplaire de la Serpe d'or et un exemplaire de Tintin au Tibet ;*
- *Marie Martin a emprunté un autre exemplaire de Tintin au Tibet ;*

- John Leroy est inscrit comme emprunteur, mais n'a rien emprunté pour l'instant.

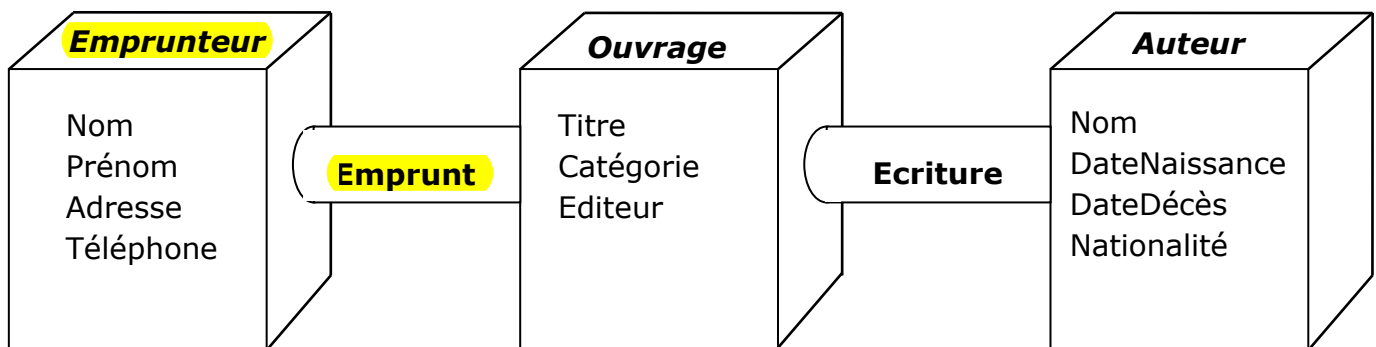
On doit pouvoir mémoriser ces emprunts et retrouver qui a emprunté quel ouvrage. On doit également pouvoir retrouver les caractéristiques des auteurs de chaque ouvrage.

Combien de types d'objet (c'est-à-dire de types d'enregistrement) faut-il prévoir ?

Lesquels ?

Quels liens prévoir entre eux ?

Quelles caractéristiques prévoir pour chaque type d'objet ?

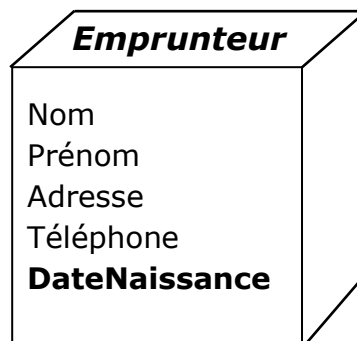


② Réduction de tarif selon le type d'emprunteur

On propose une réduction du tarif de 50% pour les plus de 60 ans et de 30% pour les moins de 25 ans.

⇒ Il suffit de prévoir une caractéristique supplémentaire pour le type d'objet *Emprunteur* qui nous permettra de calculer l'âge, à savoir la date de naissance.

NB : Quand dans les énoncé on posera des questions relatives à l'âge, il est plus que probable qu'on attende une variable *dateNaissance* et non "age". Dans ce cas, si on mettait l'age, il faudrait le changer à chaque anniversaire. En mettant la *dateNaissance*, il est changé automatiquement



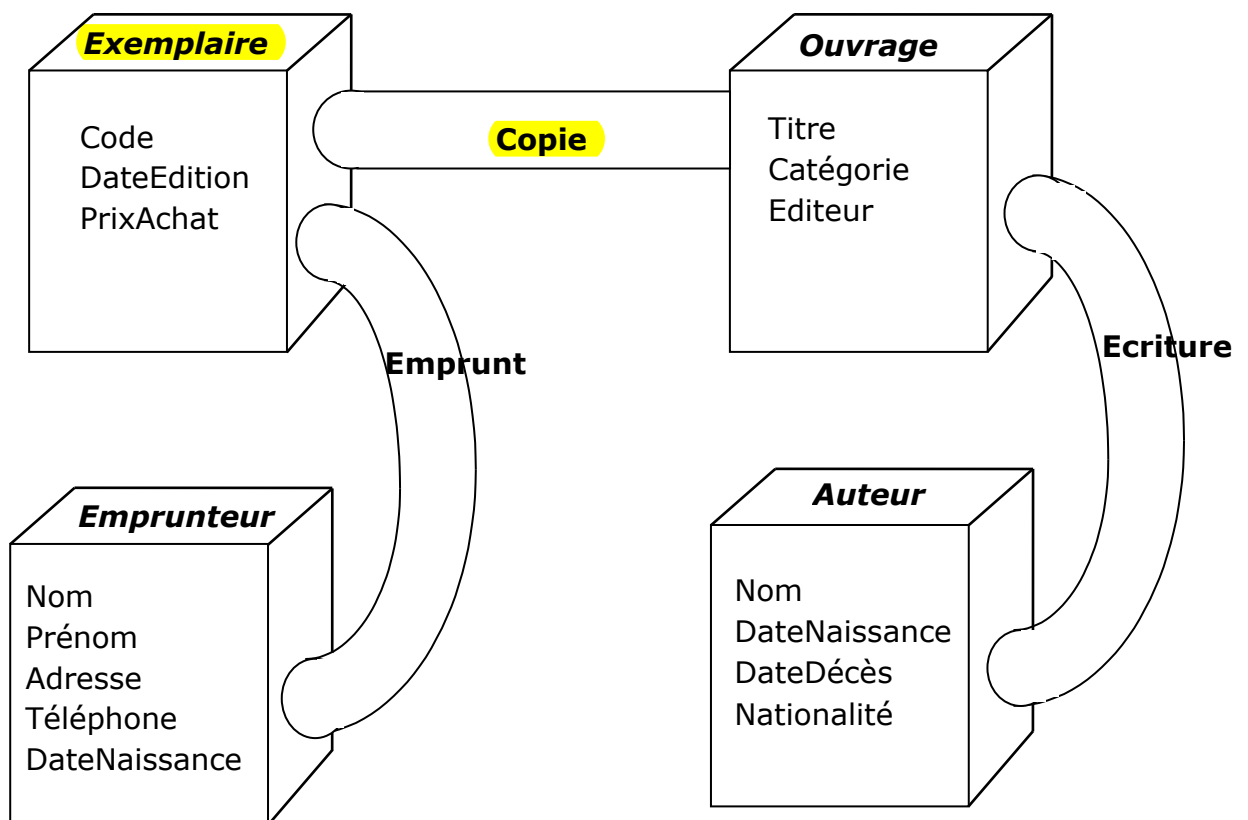
③ Retrouver la date d'édition et le prix d'achat des exemplaires empruntés

Il faut donc pouvoir identifier non plus uniquement l'ouvrage emprunté, mais **quel exemplaire est emprunté**. En effet, un même ouvrage peut être présent en plusieurs exemplaires dans la bibliothèque.

Contrairement au point ①, on ne mémorise plus quel ouvrage a emprunté un emprunteur, mais bien quel exemplaire de l'ouvrage ce dernier a emprunté. Il faut donc prévoir un type d'objet supplémentaire, à savoir **Exemplaire**.



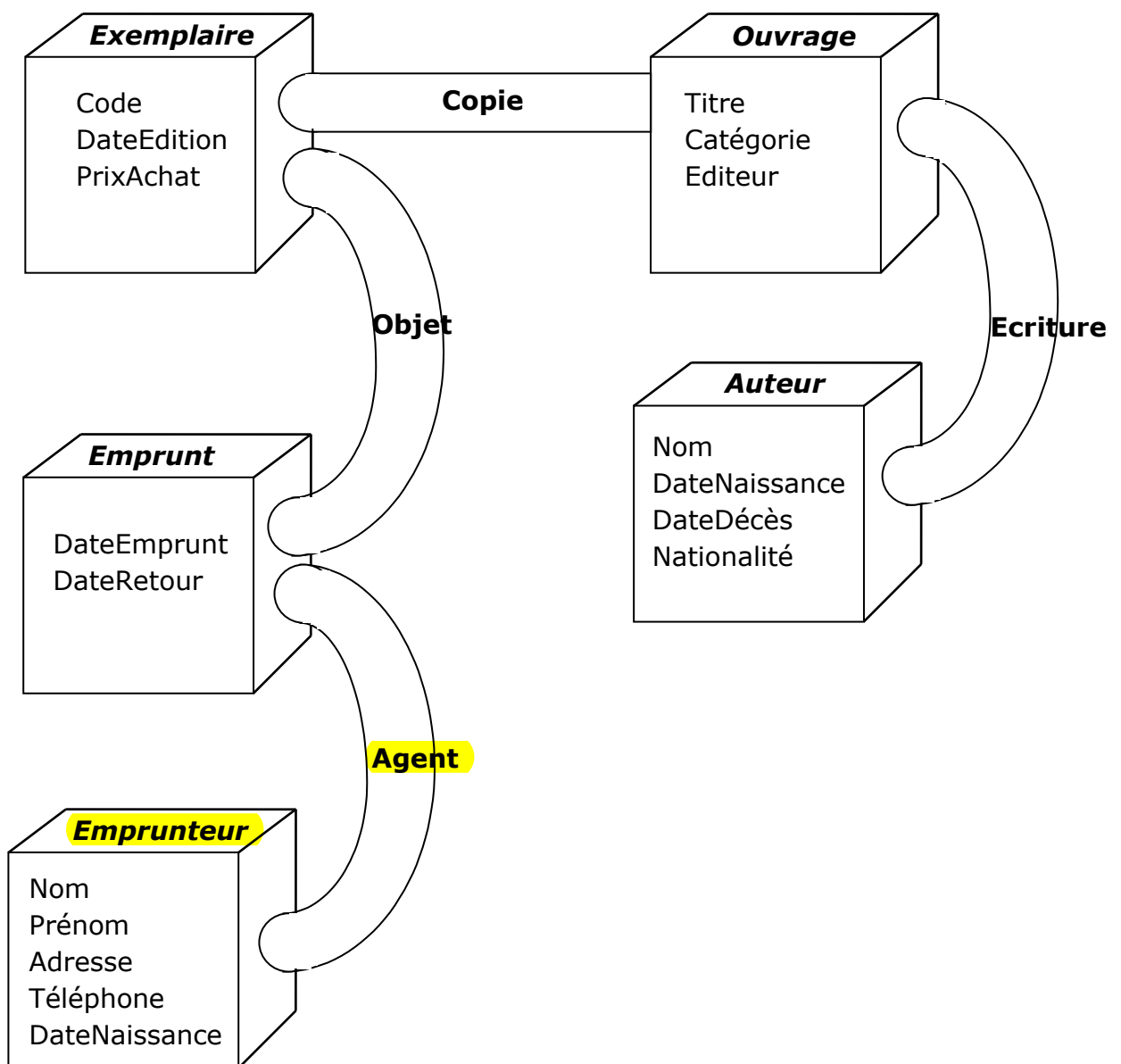
Le type d'objet *Exemplaire* est relié au type d'objet *Ouvrage*. Et le type d'objet *Emprunteur* n'est plus relié au type d'objet *Ouvrage*, mais bien au type d'objet *Exemplaire*.



④ Historique des emprunts

Supposons que, à des fins statistiques, on désire garder trace de tous les emprunts, c'est-à-dire non seulement les emprunts en cours mais également les emprunts antérieurs.

Il faut prévoir un type d'objet supplémentaire qui représentera cette notion d'emprunt. Les caractéristiques de ce nouveau type d'objet devront au moins reprendre la date d'emprunt et la date de retour. Cette dernière date permettra de distinguer les emprunts en cours des anciens emprunts : en effet, une date de retour sans valeur (inconnue) signifie qu'il s'agit d'un emprunt en cours. La date d'emprunt quant à elle permet d'identifier parmi les emprunts en cours ceux qui sont en retard.



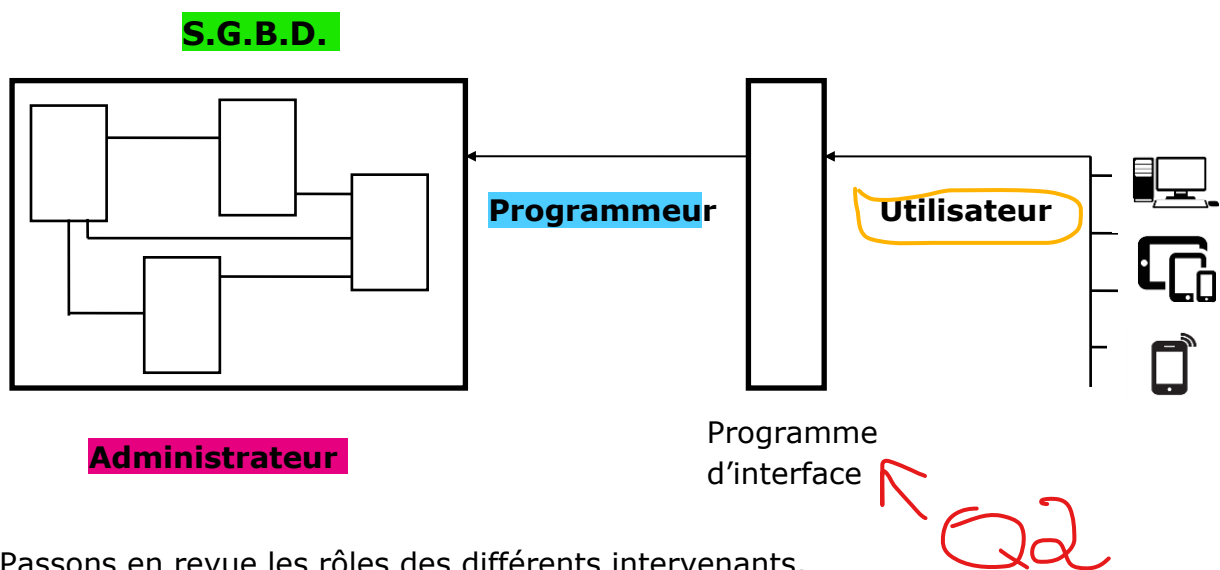
1.6 Systèmes de gestion de bases de données (S.G.B.D.)

Un S.G.B.D. ou D.B.M.S en anglais (Database Management System) est un outil permettant de gérer une base de données, à savoir, principalement :

- créer et supprimer des fichiers ;
- insérer, effacer et modifier des enregistrements dans des fichiers existants ;
- rechercher des données.

Toute base de données gérée par un S.G.B.D. doit (devrait) présenter les différentes fonctionnalités décrites au point 1.4.

L'interaction d'un S.G.B.D. avec les différents intervenants peut être illustrée comme suit :



Passons en revue les rôles des différents intervenants.

1.6.1 Le programmeur

C'est l'informaticien qui interagit directement avec le S.G.B.D. au moyen de deux types d'outils :

① Langage de programmation classique (Java, C, C#...)

Le programme d'application (interface utilisateur : écrans, menus, ...) sera écrit au moyen de langages de programmation qui spécifient le **comment faire** à travers un **algorithme** détaillant à l'ordinateur la marche à suivre.

Ces programmes dépendent du domaine d'application. Par exemple, dans le domaine bancaire, il s'agira de programme de transfert d'argent entre deux comptes...

② Langages d'exploitation de bases de données

Le langage **SQL** (**S**tructured **Q**uery **L**angage) est un langage d'accès normalisé aux bases de données relationnelles. Ce n'est pas un langage de programmation à proprement parlé mais un langage destiné à gérer et à accéder à une base de données relationnelles.

SQL peut être découpé en 4 sous-langages :

- **Data Query Language (DQL)** : langage de requêtes d'accès en lecture exprimées sous forme **déclarative** en spécifiant les critères de recherche
- **Data Definition Language (DDL)** : création et modification de schémas
- **Data Control Language (DCL)** : permissions d'accès...
- **Data Manipulation Language (DML)** : insertion, modification et suppression de données

1.6.2 L'utilisateur

Il s'agit de non informaticiens qui vont effectuer des opérations élémentaires de routine sur la B.D. via différents appareils (PC, tablette, smartphone...). Ils sont donc utilisateurs des programmes d'interface créés par les programmeurs.

Exemple : Effectuer un versement électronique

L'utilisateur a accès à une partie restreinte de la B.D.

Exemple : Le gérant de la banque a accès aux informations concernant tous les clients de l'agence et le client aux informations relatives uniquement à son compte.

La sécurité et la confidentialité des données sont assurées par des codes d'accès

Exemple : Carte bancaire et code pour retrait au distributeur

1.6.3 L'administrateur

L'administrateur, ou Database Administrator en anglais (DBA), est la personne responsable de l'ensemble du système.

Ses fonctions sont les suivantes :

① **Création de la structure originale de la B.D. et organisation des fichiers**

Via le **Data Definition Language (DDL)** (cf *create table*, *drop table* en SQL)

② **Modification de la structure de la B.D.**

Exemple : ajout/suppression d'un champ dans un fichier

Egalement via le **Data Definition Language** (cf *alter table* en SQL)

③ **Gestion des accès**

C'est l'administrateur qui donne les droits d'accès à la B.D. aux programmeurs et utilisateurs finaux.

④ **Backup et entretien de la B.D.**

1.6.4 **Le S.G.B.D.**

Le système de gestion de bases de données s'occupe de la gestion physique des fichiers. Ses fonctions sont les suivantes :

① **Accès optimal à toute donnée**

Le S.G.B.D. tente de répondre aux demandes des utilisateurs, en recherchant le chemin optimal.

② **Traitement simultané des données**

Les données sont partagées par plusieurs utilisateurs. La gestion des accès concurrents est à la charge du S.G.B.D. La notion de transaction en SQL permet entre autres de gérer en partie cette problématique.

③ Validité et cohérence des données

Les contraintes éventuelles définies par l'administrateur (à la demande des concepteurs de la B.D.) doivent être vérifiées à tout moment.

Exemples de contraintes :

- *Si l'attribut nommé Etat d'un exemplaire d'ouvrage a pour valeur "en réparation", l'exemplaire correspondant ne peut être emprunté*
- *L'âge d'un étudiant ne peut être inférieur à 17 ans*
- *Aucun compte ne peut avoir un solde inférieur à -100*

C'est le S.G.B.D. qui se charge de les vérifier à chaque modification des données.
Toute demande de modification de données qui ne satisferait pas ces contraintes est rejetée.

A noter qu'il s'agit d'un choix d'architecture. Dans certaines architectures, on préfère que la vérification des contraintes soit prise en charge par l'applicatif et non déléguée au S.G.B.D.

④ Sécurité des données

L'administrateur définit les droits des utilisateurs. A chaque demande d'accès (en lecture ou en écriture) de la part d'un utilisateur, le S.G.B.D. vérifie ses droits.

⑤ Sauvegarde et récupération

En cas de panne ou d'erreur (logicielle ou matérielle), le S.G.B.D. doit pouvoir garantir de restaurer les données dans un état **antérieur cohérent**.

En cas de panne au milieu d'une transaction, le S.G.B.D. doit pouvoir annuler la transaction et donc "remettre" la B.D. dans l'état (cohérent) précédant la transaction.

Partie 2

Analyse conceptuelle

2 Le schéma conceptuel

2.1 Introduction

Face à une application à créer, il faut en établir les spécifications exactes, c'est-à-dire définir le cahier des charges, à partir, entre autres, de l'étude du domaine d'application et de l'interview des utilisateurs.

L'analyse d'une application peut être divisée en deux parties majeures : l'analyse des traitements et l'analyse des données.

La conception d'une base de données se fera, quant à elle, en plusieurs étapes (cf. Figure 1) :

- 1 • Production d'un schéma **conceptuel**
⇒ Définition des concepts et leurs liens

- 2 • Production d'un schéma **logique**
⇒ Définition d'un schéma conforme au S.G.B.D. choisi

Exemple : Si le S.G.B.D. choisi est de type relationnel → transposition des concepts et de leurs liens en tables

Il serait intéressant de disposer d'un outil permettant de construire des schémas de données indépendamment de tout S.G.B.D. Ce type de schéma, appelé **schéma conceptuel**, devrait mettre en évidence les concepts importants et leurs relations les uns avec les autres. Un schéma conceptuel sert de support commun aux différents intervenants : analystes, administrateurs de B.D., programmeurs et, dans une moindre mesure, aux utilisateurs interviewés.

Un schéma conceptuel peut être représenté sous forme d'un **diagramme entités-associations** ou Entity Relationship Diagram (ERD) en anglais.

Un tel diagramme permet d'exprimer la sémantique des données via les notions **d'entité, d'association, d'attribut** et le **mécanisme des contraintes d'intégrité**.

Pour rappel, la notion de donnée est différente de la notion d'information :

- **Donnée** : représentation (enregistrement) codée des propriétés d'un concept, d'un objet, d'un fait.
- **Information** : signification (potentielle) attachée à une donnée.

La **sémantique** est le sens, la signification attribuée aux données.

Le but d'un schéma conceptuel est de représenter facilement, sous forme d'un schéma plus lisible, les objets au sens large et leurs relations (liens, associations).

Un tel schéma peut également servir de documentation d'une B.D. en vue de sa maintenance.

Il sera traduisible facilement en un schéma logique conforme à un type de S.G.B.D. (relationnel ou autre).

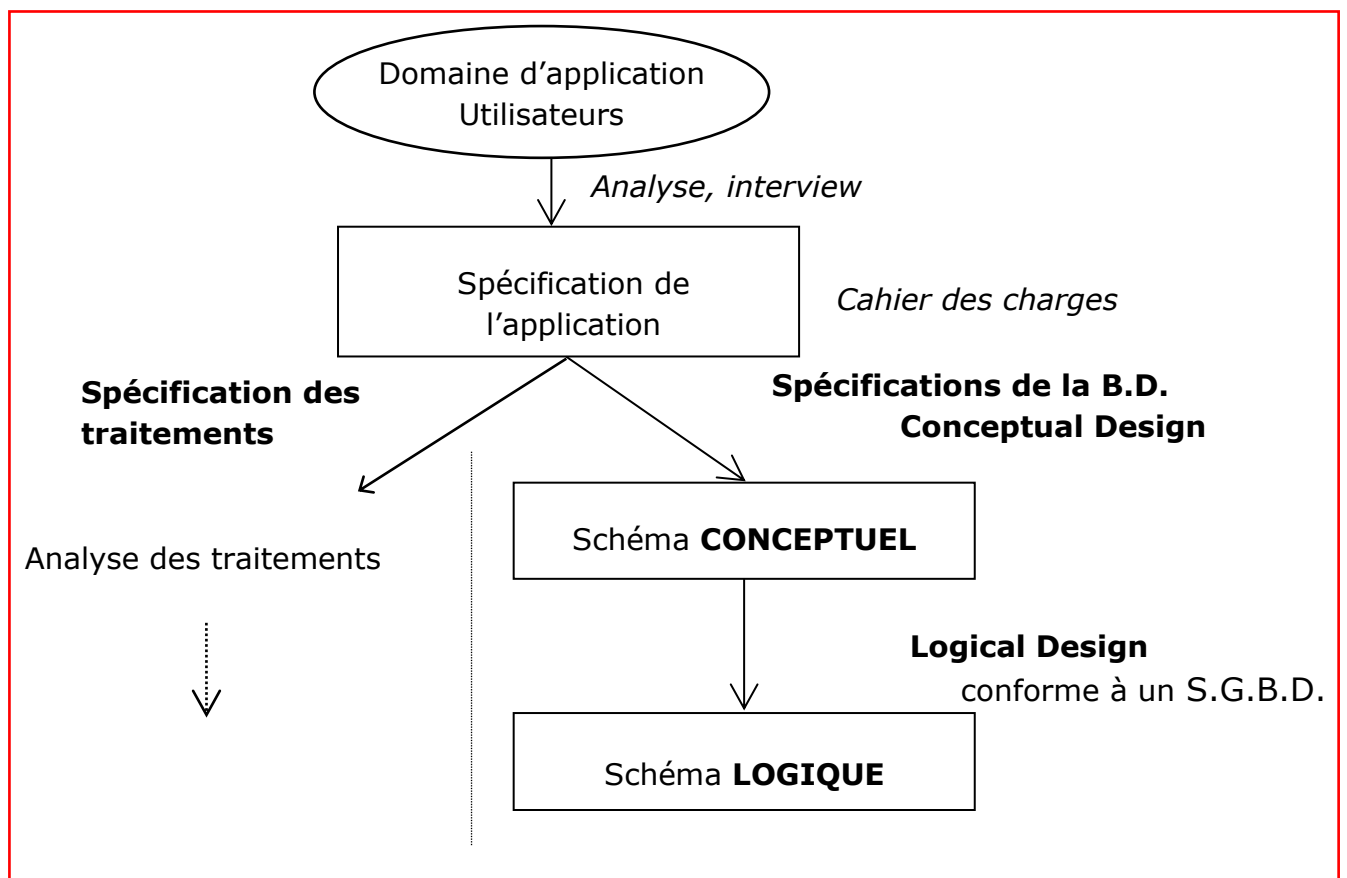


Figure 1 : Analyse d'une application

2.2 Entités

Il s'agit du concept de base du diagramme entités-associations.

Une entité est une chose qui existe dans le monde réel, à propos de laquelle on veut enregistrer des informations.

Une entité n'existe que par rapport à un individu (l'analyste) qui la considère comme importante et donc digne d'être représentée.

Une occurrence d'entité peut aussi bien représenter une chose **concrète**, physique (ex : Dupond, Tintin, mon bic, l'école (bâtiment), la voiture de M. Leroy...) qu'une chose **abstraite** (ex : L'IESN, la justice...).

Une occurrence d'entité sera caractérisée par des attributs et des valeurs d'attributs.

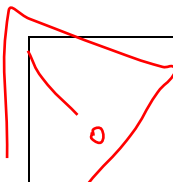
Exemple d'employé

- Nom : Dupond
- Adresse : 32, rue de Fer, 5000 Namur
- DateNaissance : 02/12/1989
- Telephone : 081 / 22.32.42

Dans l'étape de conception d'une B.D., on ne s'intéresse pas aux **éléments individuels**, aux individus en particulier, aux occurrences d'entité, mais on s'intéresse aux **types**.

Si l'on trace un parallèle avec la programmation orientée objet, on ne s'intéresse pas aux objets mais on s'intéresse aux **classes**.

Dans l'exemple, on ne s'intéresse pas aux occurrences d'employés, mais au type Employé. Tous les employés possèdent les mêmes attributs, mais chaque employé a ses propres valeurs d'attributs.



Une entité est définie par

- un nom ;
- une liste d'attributs.

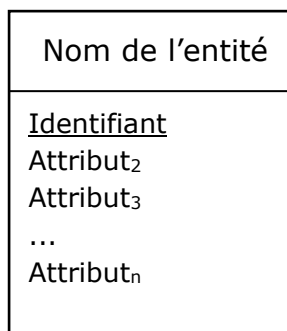
Graphiquement, une entité sera représentée par une "boîte" reprenant les différents attributs, surmontée de son **nom**.

Clé primaire (identifiant)

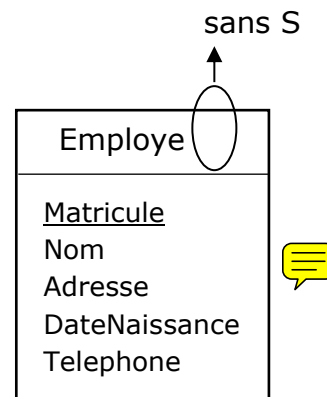
Parmi tous les attributs, certains jouent un rôle particulier, celui d'identifier chaque occurrence de l'entité. Un attribut est un **identifiant** pour une entité si **sa valeur est distincte pour chaque occurrence d'entité**.

La clé primaire ("**primary key**" en anglais) est l'identifiant principal d'une entité, en général composée d'un seul attribut. La section 2.10.1 présente des identifiants composés de plusieurs attributs.

Un attribut **identifiant** sera souligné.

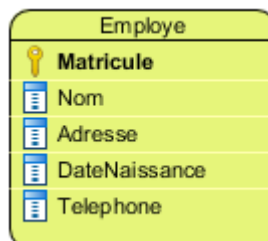


Exemple :



Certains environnements ou outils représentent d'ailleurs la clé primaire sous forme d'un symbole représentant une clé.

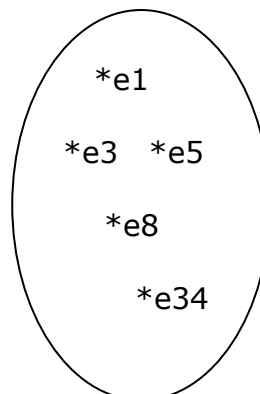
Exemple en Visual Paradigme :



⚠ Il ne faut pas confondre l'**entité** (exemple : *Employe*) et l'**ensemble de ses occurrences** (qui sera noté ***Employés*** dans la suite du cours).

Employés

Entité \neq occurrence



2.3 Associations

2.3.1 Qu'est-ce qu'une association ?

Une occurrence d'association est une correspondance, un lien, **une relation entre deux occurrences d'entités où chacune assume un rôle donné.**

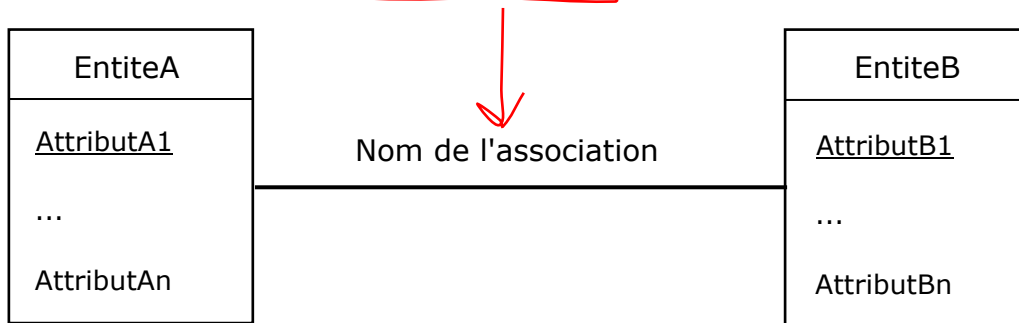
Exemple : Consultation du docteur Ledoc par le patient Jaimal

Rôles :

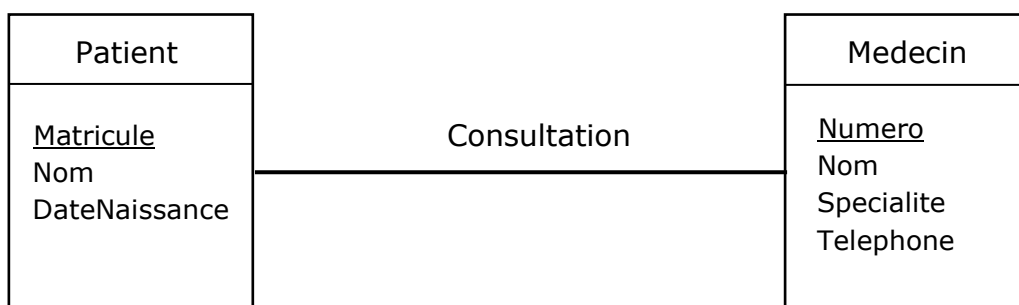
- *Jaimal* **est** le patient **ausculté**
- *Le docteur Ledoc* **ausculte** *Jaimal*

Comme pour les occurrences d'entités, on ne s'intéresse pas aux occurrences d'associations particulières entre des occurrences d'entités particulières. Dans l'étape de conception d'une base de données, on s'intéresse aux **associations entre entités**.

Graphiquement, une association sera représentée par un lien reliant les entités concernées. On donnera un nom à l'association.



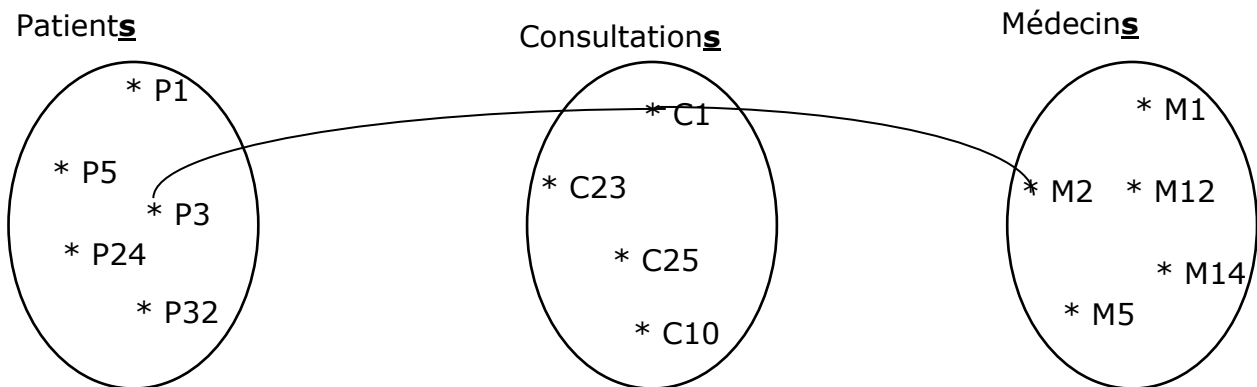
Le schéma entités-associations correspondant à l'exemple est :



Un Patient est représenté par un Matricule et un Médecin par un Numero. Les 2 sont associés entre eu par une Consultation

L'**ensemble de toutes les occurrences** d'une association forme un ensemble mathématique au même titre que l'ensemble des occurrences d'une entité.

Reprenons l'exemple



Une occurrence particulière de consultation (soit l'occurrence C1) est un lien entre un patient particulier (P3) et un médecin particulier (M2).

Quelles sont exactement les relations entre ces ensembles ?

A combien de relations (liens) au minimum et au maximum participe un élément donné ?

• Un élément de l'ensemble des *Patients* **peut**-il avoir **plusieurs** liens avec des éléments de l'ensemble des *Consultations* ?

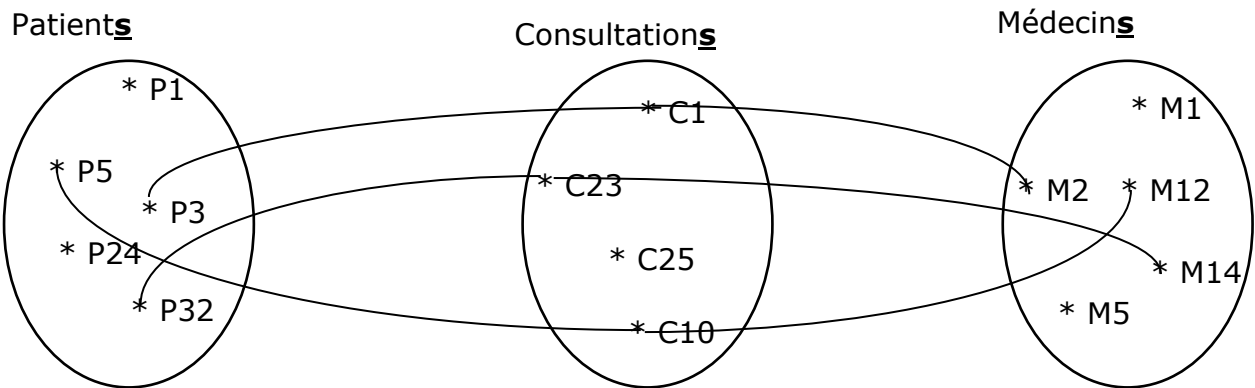
• Un élément de l'ensemble des *Patients* **doit**-il obligatoirement avoir **au moins un** lien avec un élément de l'ensemble *Consultations* ?

• Un élément de l'ensemble des *Médecins* **peut**-il avoir **plusieurs** liens avec des éléments de l'ensemble des *Consultations* ?

• Un élément de l'ensemble des *Médecins* **doit**-il obligatoirement avoir **au moins un** lien avec un élément de l'ensemble *Consultations* ?

• Combien de liens un élément de l'ensemble des *Consultations* a-t-il avec l'ensemble des *Patients* et avec l'ensemble des *Médecins* ?

💣 Une occurrence d'association est toujours reliée à **une et une seule occurrence** de chaque entité associée.



Une occurrence de *Consultation* ne peut être reliée qu'à une seule occurrence de *Patient* et à une seule occurrence de *Medecin*. En effet, un patient donné consulte un médecin particulier ; par exemple, le patient *Jaimal* qui consulte le docteur *Ledoc*.

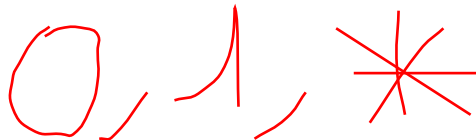
Mais, une occurrence particulière de *Patient* peut être reliée à **plusieurs** occurrences de *Consultation* : un patient peut consulter plusieurs médecins.

De même, une occurrence particulière de *Medecin* peut être reliée à **plusieurs** occurrences de *Consultation* : un médecin peut être consulté par plusieurs patients.

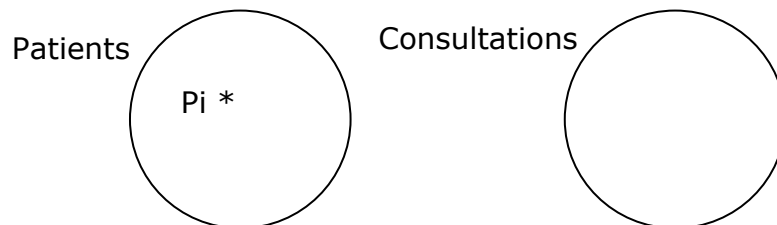
Les questions intéressantes à se poser sont donc les suivantes :

A) A combien d'occurrences de *Consultation* peut être reliée une occurrence de *Patient* ?

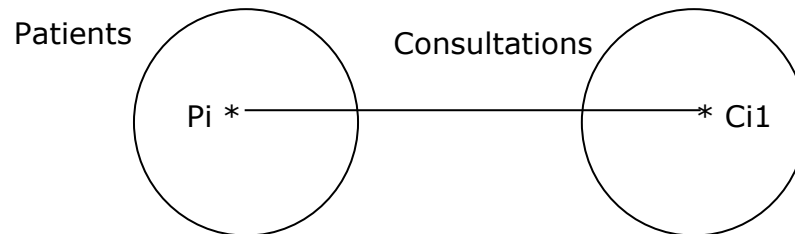
Trois possibilités : 0, 1 ou N.



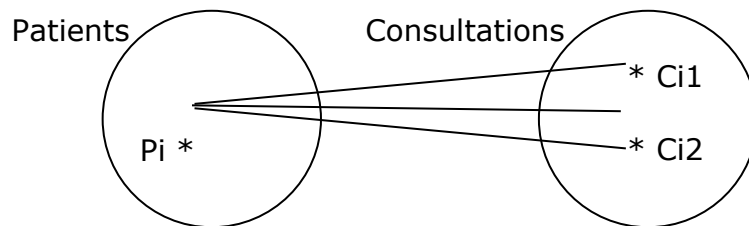
0 : si le patient n'a (encore) consulté aucun médecin, il n'est relié à aucune consultation.



1 : si le patient n'a consulté qu'un seul médecin, il est relié à une et une seule consultation



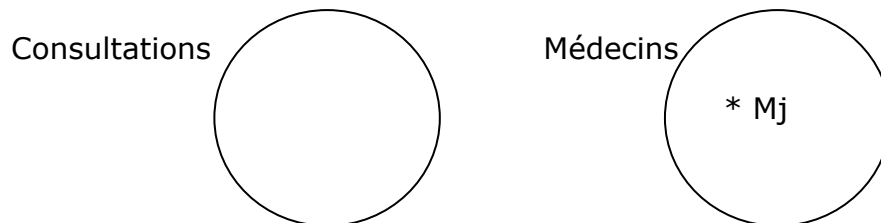
N : si le patient a consulté plusieurs médecins : il est relié à plusieurs consultations.



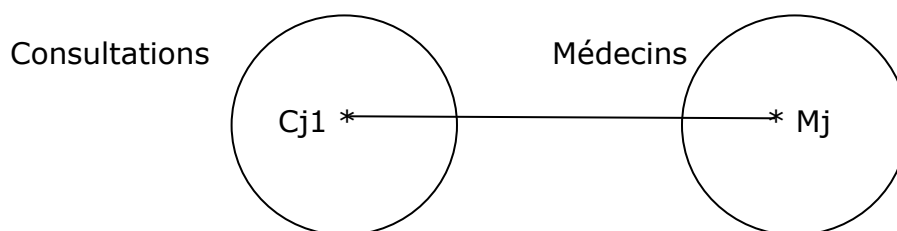
B) A combien d'occurrences de Consultation peut être reliée une occurrence de Medecin ?

Trois possibilités également : 0, 1 ou N.

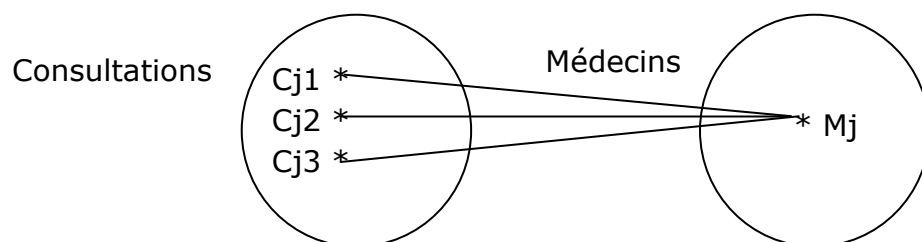
0 : si le médecin n'a toujours pas été consulté.



1 : si le médecin n'a donné qu'une seule consultation.



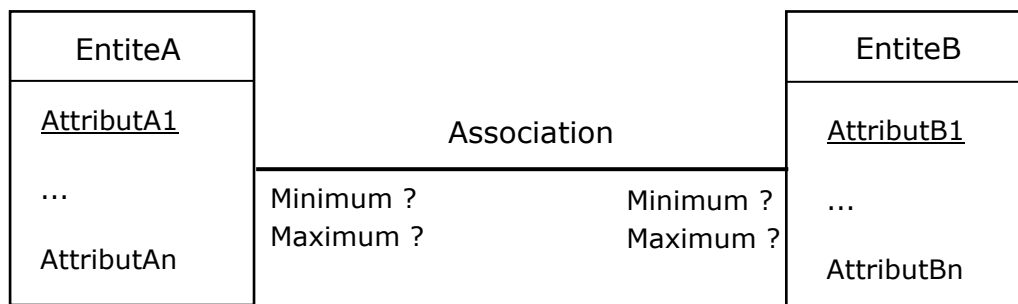
N : si le médecin a donné plusieurs consultations.



N'importe quelle occurrence d'une entité peut avoir au minimum **Min** et au maximum **Max** liens avec des occurrences d'associations.

On parlera de **cardinalités** ou **connectivités minimum** et cardinalités ou connectivités **maximum**.

Des cardinalités sont associées à une entité pour chaque association.



2.3.2 Cardinalités maximales = obligatoire

Une association est une **relation** entre deux entités.

Les cardinalités maximales d'une association dépendent du **type de cette relation**. Il y a trois types de relation :

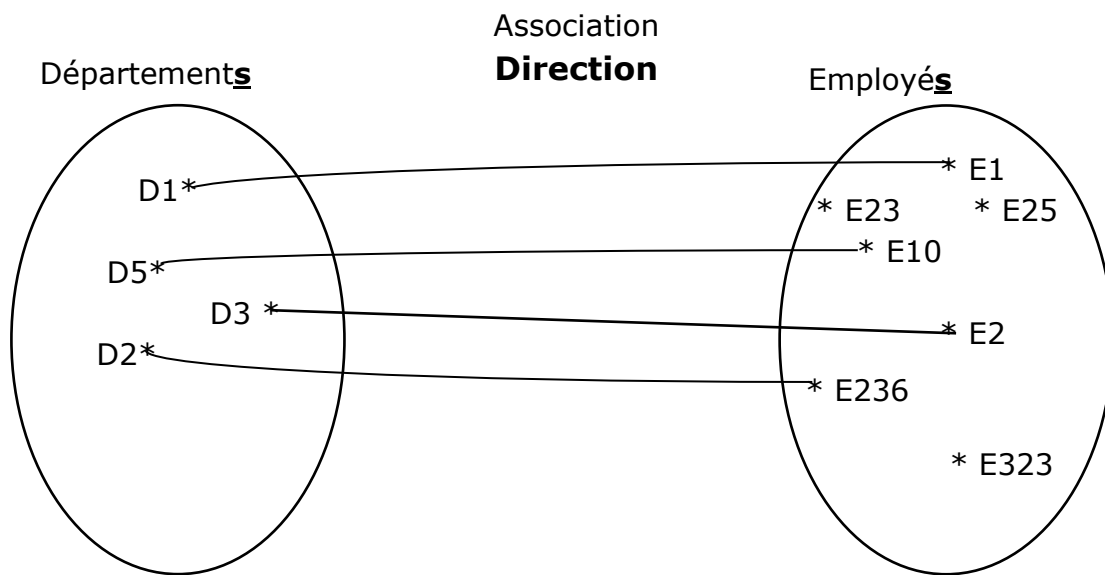
- **1 à 1**
- **1 à N (ou 1 à plusieurs ou 1 à *)**
- **N à N (ou plusieurs à plusieurs ou * à *)**

La notion de **cardinalités** concerne les **associations**, tandis que la notion de **type de relation** concerne les **ensembles**.

La notion de type de relation entre ensembles n'intervient que dans le calcul des cardinalités **maximum** !

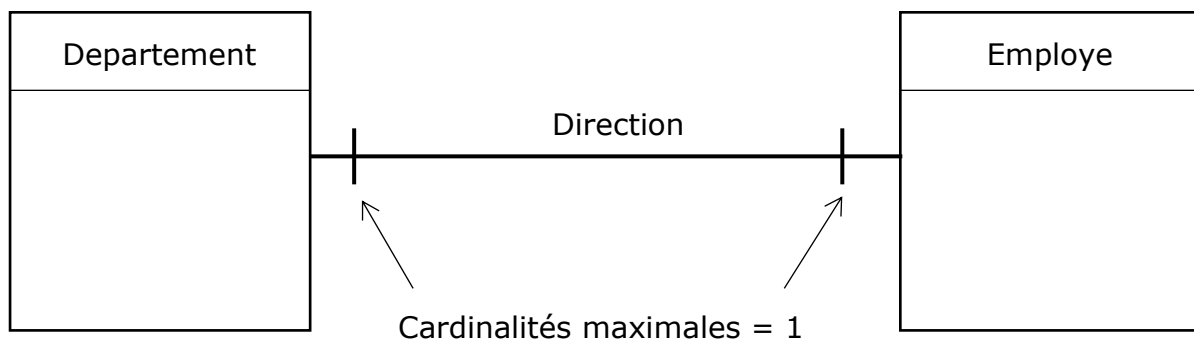
Les cardinalités maximales ne peuvent être que de **1 ou N (plusieurs)**.

A. Relation de type 1 à 1

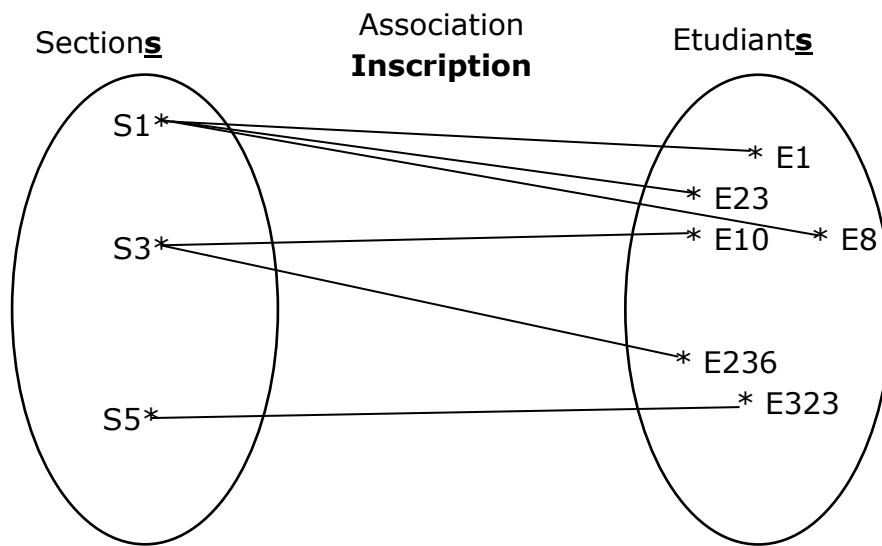


Au maximum :

- Un département a **au plus un** employé qui est directeur
- Un employé est directeur d'**au plus un** département

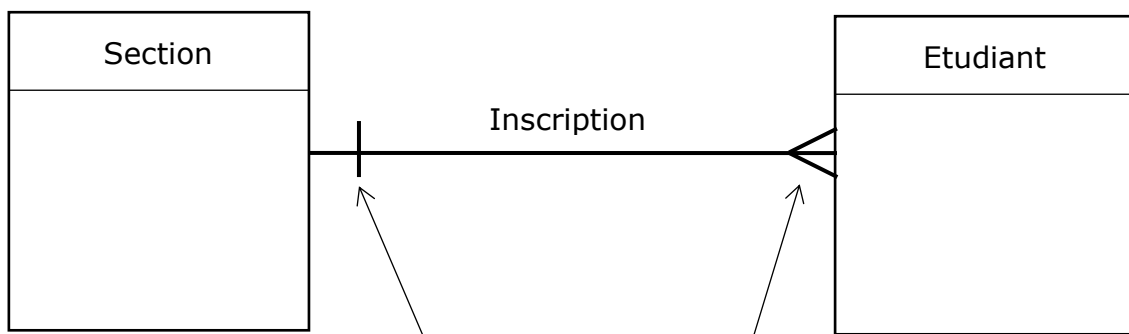


B. Relation de type 1 à N



Au maximum :

- **Plusieurs** étudiants peuvent s'inscrire dans **une même** section
- Un étudiant est inscrit dans **au plus une** section

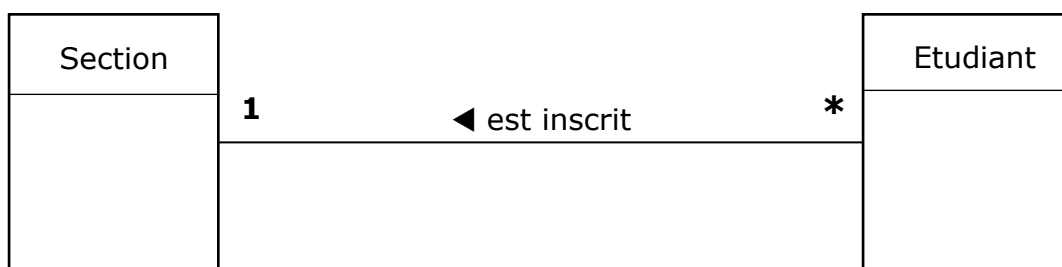


Cardinalité maximale = 1

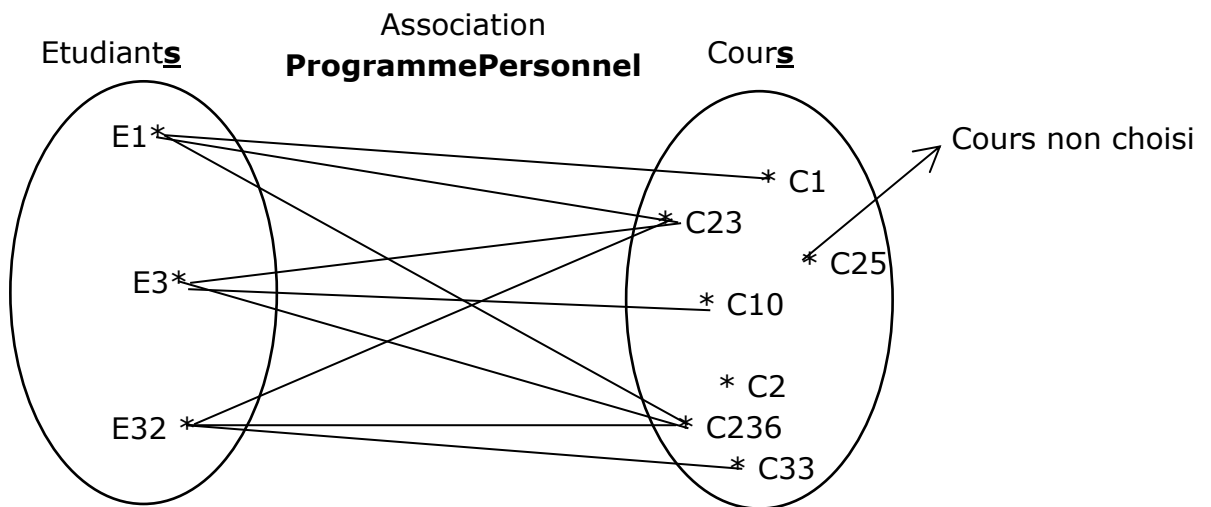
Cardinalité maximale = plusieurs

Cette notation est à mettre en parallèle avec les cardinalités/multiplicités du **diagramme de classes** des schémas UML.

L'exemple ci-dessus est modélisé en diagramme de classes sous le schéma suivant :

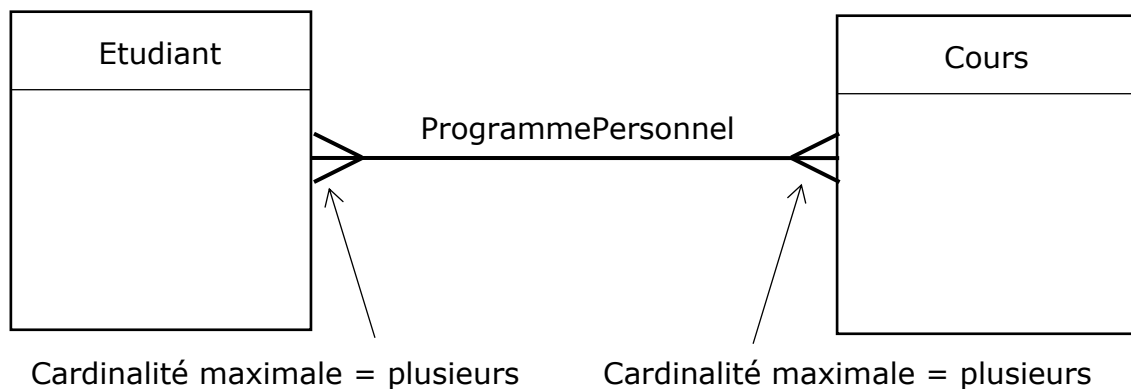


C. Relation de type N à N



Au maximum :

- Un étudiant peut choisir **plusieurs** cours dans son PAE
- Un cours peut être choisi par **plusieurs** étudiants



2.3.3 Cardinalités minimales = Facultative

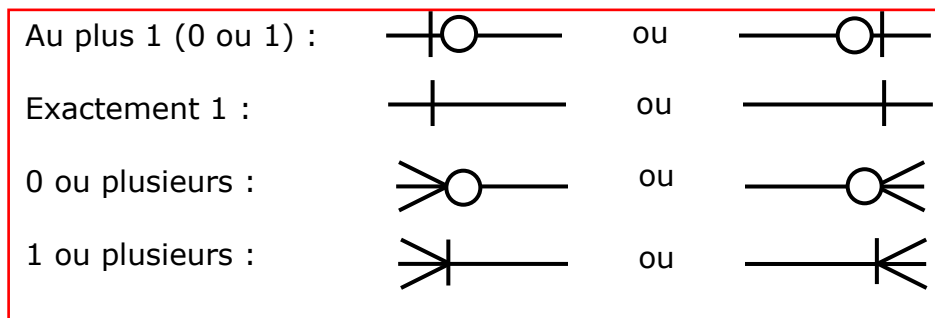
Les cardinalités minimales expriment le fait qu'une association est **facultative** ("peut") ou **obligatoire** ("doit").

La cardinalité minimale d'une association **facultative** est égale à **0**.

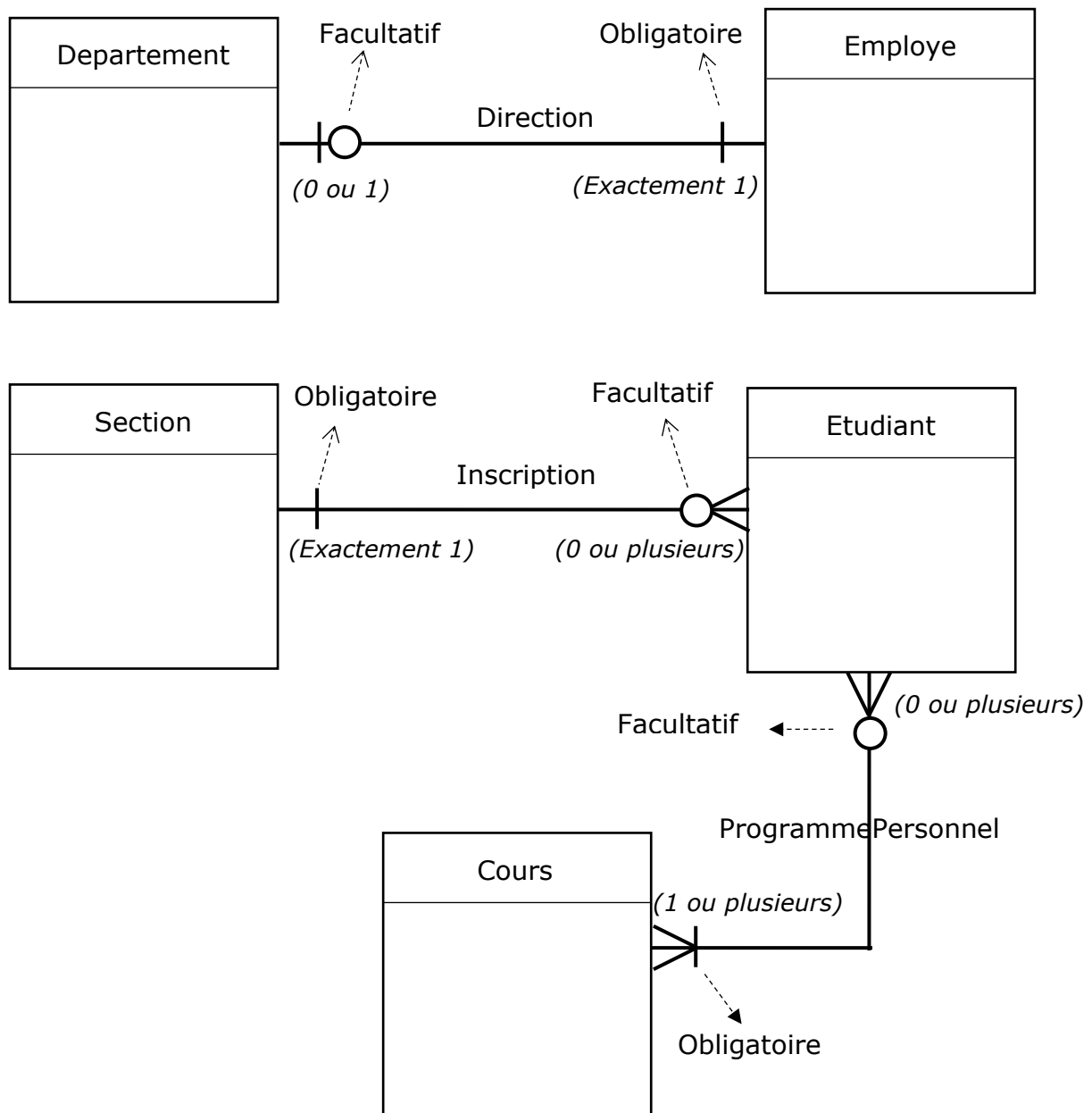
La cardinalité minimale d'une association **obligatoire** est égale à **1**.

Il s'agit de contraintes sur le **nombre minimum** d'entités associées.

Combinaisons possibles des cardinalités minimales et maximales :



Exemples



Interprétation des cardinalités minimales :

Direction

- Un département est **toujours** sous la direction d'**un et un seul** employé.
- Un employé **peut ne pas** être directeur.

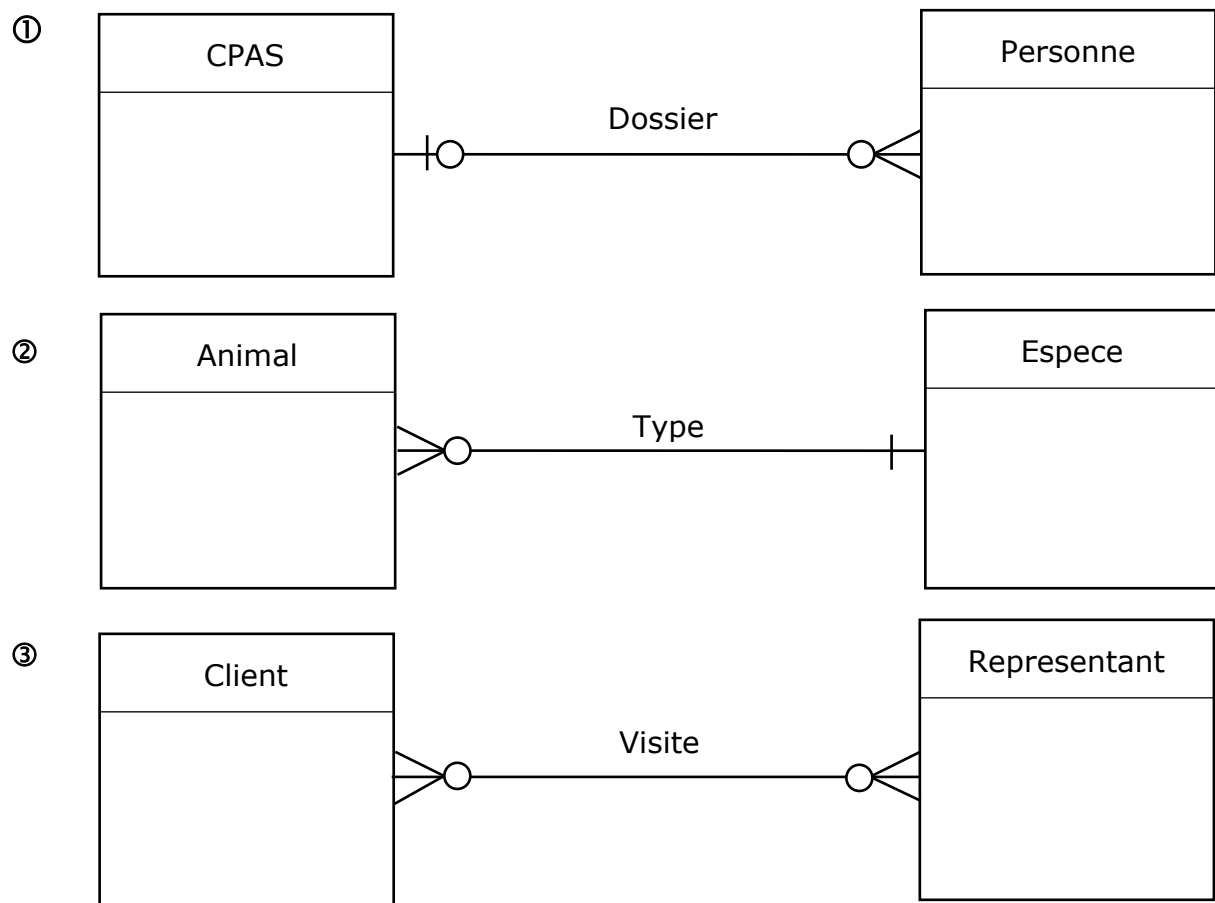
Inscription

- Une section **peut** n'avoir **aucun** étudiant (par exemple, une nouvelle section qui vient de s'ouvrir).
- Un étudiant **doit** être inscrit dans une section.

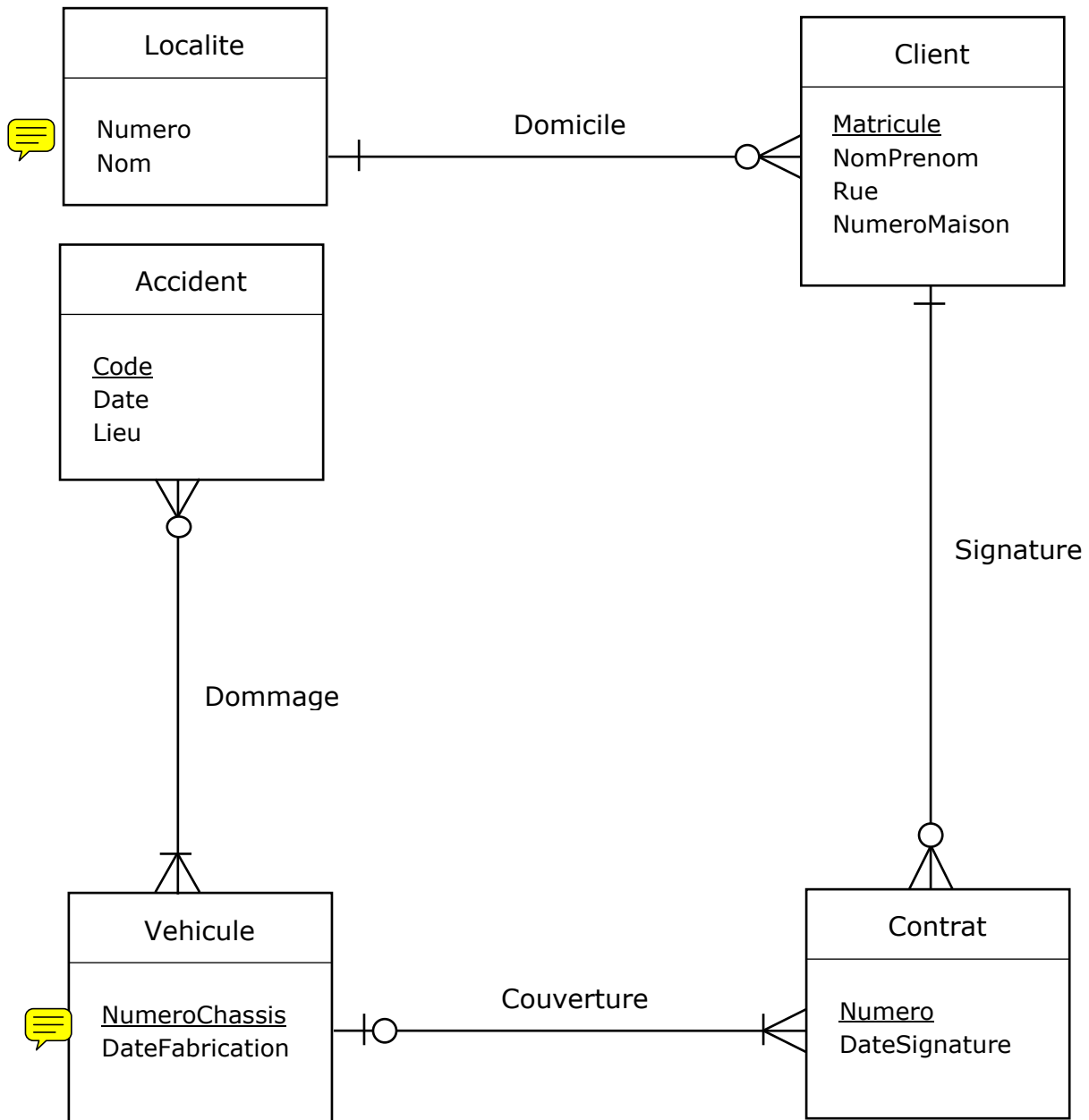
ProgrammePersonnel

- Un étudiant **doit** choisir **au moins un** cours.
- Un cours **peut** n'avoir été choisi par **aucun** étudiant.

2.3.4 Exemples



④ Une compagnie d'assurances demande à un de ses informaticiens de créer le schéma conceptuel de sa base de données. Ne sont stockées dans la base de données que les informations nécessaires à la compagnie d'assurances.



Interprétation :

- En Belgique, le code postal seul n'est pas identifiant (exemples : 5020 Dausseulx, 5020 Malonne, 5020 Vedrin). De plus, le même nom est donné à plusieurs localités (exemples : 6230 Buzet dans la province du Hainaut et 5530 Buzet dans la province de Namur). Une localité sera donc identifiée par un numéro unique. A noter que la combinaison du code postal et du nom de la localité forme un identifiant naturel (cf. section 2.10.1. Identifiant composé d'attributs).
- Tout client est domicilié dans une et une seule localité (cardinalités 1-1).
- Une localité reprise dans la base de données n'est pas forcément la localité du domicile d'au moins un client. C'est le cas par exemple si on importe dans la B.D. le répertoire de toutes les localités de Belgique (cardinalités 0-N).
- Un client peut avoir signé plusieurs contrats (cardinalités 0-N).
- Tout contrat est signé par un et un seul client (cardinalités 1-1).
- Un contrat ne couvre pas forcément un véhicule ; la compagnie ne s'est pas spécialisée exclusivement dans l'assurance automobile. Un même contrat ne peut couvrir qu'un véhicule à la fois. Les cardinalités sont donc 0-1.
- Si un véhicule est repris dans la base de données de la compagnie, c'est qu'il fait l'objet d'au moins un contrat d'assurance. Un même véhicule peut faire l'objet de plusieurs contrats (par exemple, l'assurance vol fait l'objet d'un contrat indépendant de la couverture classique). Les cardinalités sont donc 1-N.
- Un véhicule peut être impliqué dans plusieurs accidents. Il peut n'avoir été impliqué dans aucun accident (cardinalités 0-N).
- Un accident répertorié par l'agence implique au moins un véhicule enregistré dans la base de données, et pourrait malheureusement en impliquer plusieurs (cardinalités 1-N)

2.4 Construction d'un schéma conceptuel

Le schéma conceptuel est élaboré par les analystes. Ceux-ci se basent entre autres sur les interviews réalisées avec le client demandeur de l'application à réaliser, afin de maîtriser les concepts du domaine d'application et leurs liens. On peut voir une interview comme étant l'énoncé d'une succession de phrases ou propositions émises par le client à propos de son domaine d'application.

Malheureusement, ces propositions peuvent être **incomplètes, redondantes**, voire mutuellement **contradictaires** ou carrément **fausses**.

2.4.1 Décomposition de l'énoncé

- Quand c'est possible, l'énoncé doit être décomposé en propositions élémentaires du type :

sujet - verbe - complément

Exemples

- *Tout étudiant a une date de naissance.*
- *Une commande est passée par un client.*
- *Un véhicule appartient à une personne.*

Ce type de proposition élémentaire affirme l'existence de **deux concepts** (le sujet et le complément) et **un lien** (le verbe).

Exemple : Tout étudiant a une date de naissance :

→ **concepts** : *étudiant* et *date de naissance*

→ **lien** : *possession*

- Il existe cependant d'autres types de propositions élémentaires.

Exemples :

- *Il existe des professeurs.*
- *On organise des examens.*

Ce type de proposition élémentaire affirme l'existence d'**un seul concept sans lien**.

- Il faut parfois reformuler certaines phrases complexes.

Exemple 1 : Un étudiant est identifié par un matricule et est caractérisé par un nom et une date de naissance.

Il faut éclater cette proposition en trois phrases simples :

- ① *Un étudiant est identifié par un matricule.*
- ② *Un étudiant est caractérisé par un nom.*
- ③ *Un étudiant est caractérisé par une date de naissance.*

Chacune de ces propositions affirme l'existence de **deux concepts et un lien**.

Exemple 2 : La plaque minéralogique de chaque véhicule contient ...

Il faut scinder cette proposition en deux :

- ① *Tout véhicule a une plaque minéralogique*
- ② *La plaque minéralogique contient ...*

- Il faut également expliciter les raccourcis du langage que sont les pronoms possessifs.

Exemples

- *Il peut acheter un produit (il = client).*
- *Sa localité... (= le client a une localité et la localité...).*

- Pour les propositions de type "sujet verbe complément", c'est-à-dire "**A verbe B**", qui affirment l'existence de deux concepts et un lien, on cherche :

- pour un exemplaire de A, **combien a-t-on d'exemplaires** de B au minimum et maximum ?

- pour un exemplaire de B, **combien a-t-on d'exemplaires** de A au minimum et maximum ?

Exemples

- *Un étudiant est inscrit dans une seule section.*
- *Un cours peut être choisi par plusieurs étudiants.*

N.B. Les réponses sont parfois dans les phrases à travers des mots comme :

- **tout, certains, chaque**
- **pouvoir, devoir**
- **au moins un, un seul, au plus un**
- **des, les...**

Propositions générales et particulières

Il est fondamental pour un concepteur de bases de données de faire la distinction entre des propositions générales et des propositions particulières.

Exemples

- *Toute personne a un nom*
 - ✎ Proposition **générale** (niveau **type** ou **classe** d'objets)
- *Je m'appelle Arnaud Lebel*
 - ✎ Proposition **particulière** (niveau **occurrence** d'objets)

Il s'agit d'une instance, d'une occurrence, d'un exemple.

Dans un schéma de base de données, il ne faut pas représenter les propositions particulières.

Il faut seulement représenter les propriétés, les caractéristiques et les liens des **types**.

Mais les propositions particulières sont cependant utiles, car elles peuvent suggérer (par **généralisation à partir d'exemples**) une caractéristique ou un lien de la classe/type correspondante.

2.4.2 Représentation d'une proposition

On a identifié des concepts et des liens.

Qu'est-ce que cela devient en termes d'entités et d'associations ?

A. Concepts

- Si un **concept semble important** dans le contexte du domaine d'application → **entité**.

Exemple : *Il y a des clients*

Dans la gestion d'une entreprise de services, la notion de client peut à première vue être un concept suffisamment important → **entité** *Client*

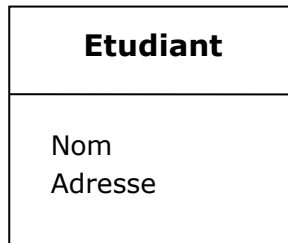
Client

- Si un concept contient **plusieurs caractéristiques** → **entité**.

Exemples

- *Tout étudiant a un nom*
- *Tout étudiant a une adresse*

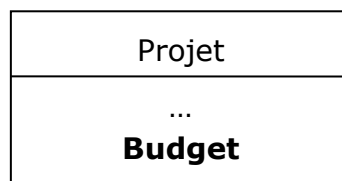
Le même concept contient deux caractéristiques → **entité** *Etudiant*



- Si c'est une **simple propriété** d'un autre concept existant → **attribut**

Exemple : Tout projet a un budget

Si l'entité *Projet* existe déjà, le concept *Budget* est un attribut de l'entité *Projet* → **attribut**



B. Liens

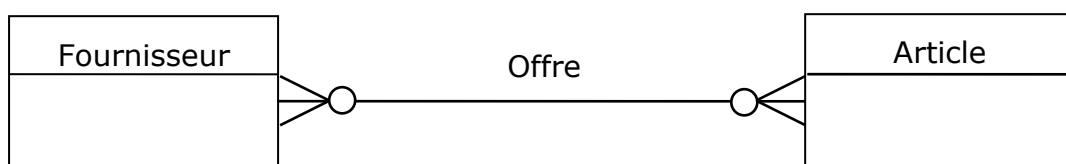
La représentation des liens dépend du type des concepts reliés.

B.1. Entre deux entités

S'il s'agit d'un lien entre deux concepts eux-mêmes représentés par **deux entités** → le lien est une **association**.

Exemple : Un fournisseur propose des articles

Si *Fournisseur* et *Article* sont déjà représentés chacun par une entité → association *Offre*



B.2. Entre une entité et un attribut

S'il s'agit d'un lien entre deux concepts, l'un étant déjà représenté par **une entité**, l'autre étant une **simple caractéristique** du premier → on crée un **nouvel attribut** qu'on affecte à l'entité.

Exemple : Tout terrain a une superficie

Si l'entité *Terrain* existe déjà, le concept *Superficie* est un attribut de l'entité *Terrain* → **attribut**

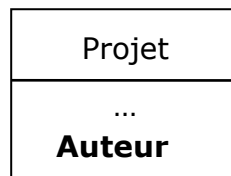


B.3. Entre deux attributs

Comment établir un lien entre un attribut déjà existant sur le schéma et un nouveau concept qui semble n'être qu'une caractéristique de cet attribut ?

→ Il faut créer une **nouvelle entité** à laquelle on affecte ces deux attributs, ainsi qu'une **nouvelle association** entre l'entité existante et cette nouvelle entité.

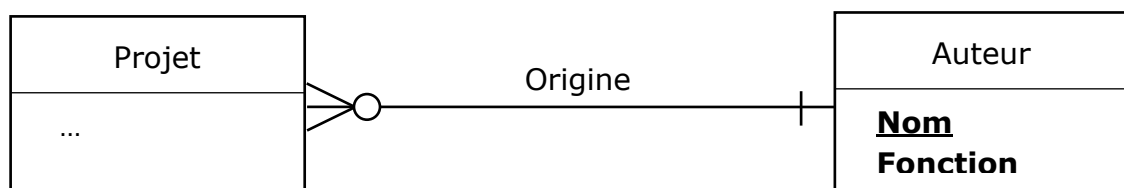
Exemple : 1^{ère} étape : A tout projet est associé le nom de son auteur



2^{ème} étape : L'auteur du projet a une fonction

"fonction" est une caractéristique supplémentaire d'*Auteur* qui est lui-même un attribut déjà existant.

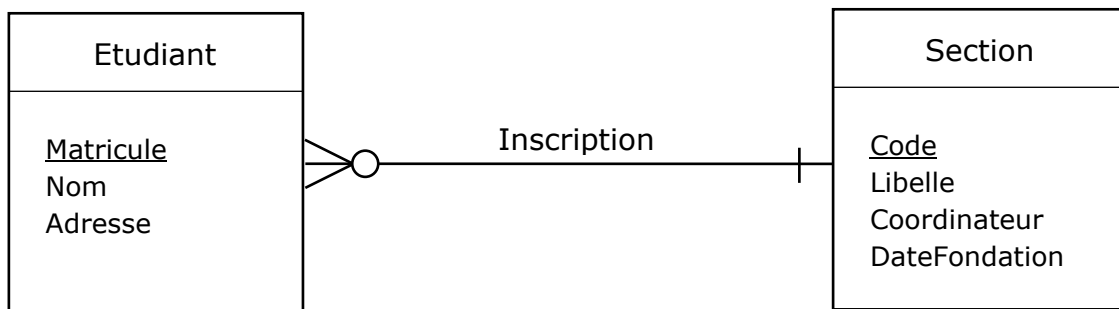
↪ On extrait l'attribut *Auteur* de l'entité *Projet*. On crée une nouvelle entité qu'on appelle *Auteur* qui aura deux attributs, un attribut identifiant (*Nom*) et un attribut *Fonction*. Une nouvelle association *Origine* est créée pour relier les entités *Projet* et *Auteur*.



2.5 Intérêt de prévoir une association plutôt qu'un attribut

On pourrait se demander quel est l'intérêt pour un analyste de prévoir une association entre deux entités plutôt que de prévoir un attribut dans une des entités "mémorisant" ainsi un lien vers l'autre entité.

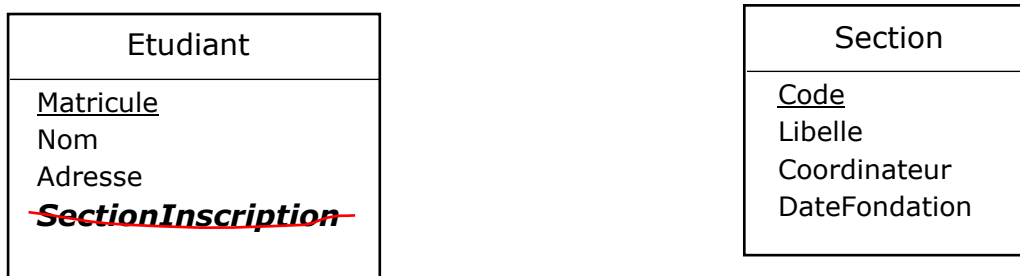
Exemple : Soit le schéma suivant



Pourquoi prévoir cette association *Inscription* plutôt qu'un simple attribut ***SectionInscription*** dans l'entité *Etudiant* ?

L'attribut ***SectionInscription*** pourrait alors prendre comme valeur n'importe quelle chaîne de caractères. Le problème, c'est qu'il pourrait aussi prendre comme valeur **un code de section inexistant !!!**

Le schéma deviendrait alors :



La différence fondamentale entre les deux schémas est que si le S.G.B.D. est prévenu de l'existence de l'association *Inscription*, il **est tenu de vérifier la cohérence de la base de données**. Par conséquent, il s'assurera qu'à chaque création (ou modification) d'un *Etudiant*, celui-ci soit relié à une section **existante**. Par contre, si cette association *Inscription* n'est pas spécifiée au S.G.B.D., celui-ci ne peut maintenir la base de données dans un état cohérent. En effet, rien n'empêchera la création/modification d'un étudiant avec l'affectation à l'attribut ***SectionInscription*** d'un code de **section inexistante**.

Une telle base de données est difficilement exploitable : lors de l'interrogation de la base de données, il n'est pas certain de pouvoir retrouver les informations concernant la section de chaque étudiant, puisque certains étudiants risquent d'être enregistrés avec un code de section erroné. Par contre, si l'association *Inscription* est déclarée, on est certain de pouvoir retrouver pour chaque étudiant les informations concernant sa section, et inversement, pour chaque section, les informations concernant les étudiants qui y sont inscrits.

En conclusion, un S.G.B.D. est un outil formidable pour l'administrateur de la base de données. En effet, bon nombre de **contraintes peuvent être placées sous la responsabilité du S.G.B.D.** Il peut ainsi effectuer toutes sortes de vérifications, et, si une contrainte est violée lors de l'accès à la base de données, refuser cet accès à l'utilisateur. A l'analyste donc de prévoir toutes ces contraintes **dès la création de la base de données**. La déclaration d'une association entre deux entités en est une.

2.6 Attributs

Un attribut est défini par un **nom**, un **type** et le **domaine des valeurs** admises.

💣 Les types et domaines de valeurs n'apparaissent pas dans le schéma entités-associations. Ils seront renseignés dans une documentation annexe.

Dans cette documentation, chaque entité et chaque association fera l'objet d'une description détaillée.

Il existe différentes classes d'attributs.

2.6.1 Identifiant ("primary key")

Un attribut est un identifiant dans une entité si sa **valeur est distincte pour chaque occurrence** de l'entité (contrainte d'unicité – "uniqueness"). Autrement dit, deux occurrences d'entités ne peuvent pas avoir la même valeur d'identifiant.

L'attribut identifiant est souligné sur le schéma entités-associations.

Exemple : Une personne est identifiée par son numéro de registre national

Personne
<u>NumeroRegistreNational</u>

2.6.2 Obligatoire ou facultatif

Un attribut est **obligatoire** si toute occurrence de l'entité **doit** avoir une valeur pour cet attribut.

Exemple : Toute personne a obligatoirement un nom

Personne
<u>NumeroRegistreNational</u> Nom

Un attribut est **facultatif** si une occurrence de l'entité **peut** ne pas avoir de valeur pour cet attribut. Un attribut **facultatif** est un attribut dont la valeur peut être inconnue (valeur null).

Un attribut peut ne pas avoir de valeur pour une occurrence d'entité particulière

- Soit parce que la valeur de cet attribut est inconnue

Exemple : numéro de téléphone, si la personne possède le téléphone mais ne désire pas communiquer son numéro

- Soit parce que l'attribut est sans signification pour cette occurrence-là

Exemple : nom d'épouse pour une personne de sexe masculin

Un attribut facultatif est spécifié sur le schéma entité-association par des cardinalités [0..1].

Exemples :

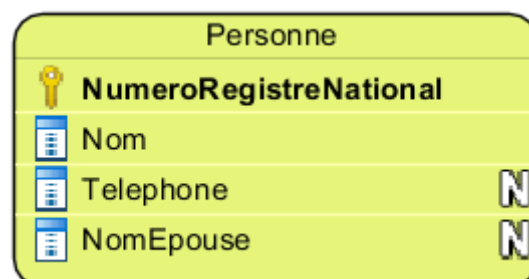
- Une personne **peut** avoir un numéro de téléphone
- Une personne **peut** avoir un nom d'épouse

Personne
<u>NumeroRegistreNational</u>
Nom
Telephone [0..1]
NomEpouse [0..1]

N.B. Un attribut **obligatoire** et **mono-valué** (le plus fréquent) a des cardinalités **[1..1]**. Par convention, on ne note alors aucune cardinalité sur le schéma entités-associations. Tout attribut apparaissant sur le schéma entités-associations **sans cardinalité** a des cardinalités **[1..1] implicites**.

A titre indicatif, les attributs facultatifs sont parfois représentés par le symbole **N** (pour nullable) dans certains environnements.

Exemple en Visual Paradigm :



2.6.3 Atomique ou décomposé

Un attribut est **atomique** ou **élémentaire** si ses valeurs sont **indivisibles**, c'est-à-dire si on ne veut ou on ne peut plus les décomposer en parties plus fines.

Exemple : Nom

Un attribut peut être **décomposé en attributs plus fins** quand on désire pouvoir accéder à chacune des parties de l'attribut. Par exemple, si l'adresse est stockée uniquement en vue d'imprimer des étiquettes à apposer sur des enveloppes à envoyer par courrier, un attribut *adresse* de type atomique suffit.

Si, par contre, des statistiques doivent pouvoir être calculées par code postal, localité, rue... , l'attribut *adresse* doit être décomposé en attributs plus fins.

Exemple :

```

  Adresse
    /  \
   Rue  Ville
    / \  / \
  NomRue Numero Boite [0..1]
                          boite est facultative
    / \
  CodePostal
  NomLocalite
  
```

Chacune de ces parties élémentaires fera l'objet d'un attribut à part entière.

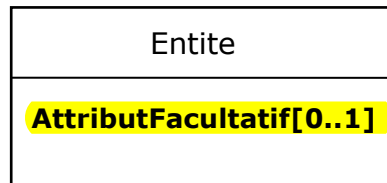
Personne
<u>NumeroRegistreNational</u> Nom Telephone [0..1] NomEpouse [0..1] NomRue Numero Boite [0..1] CodePostal NomLocalite

Ne pas confondre [0..1] et estUnBooleen et estBooleen[0..1]

2.6.4 Attribut facultatif versus attribut booléen

Pour rappel, un attribut **facultatif** est un attribut dont la valeur peut être inconnue (valeur **null**).

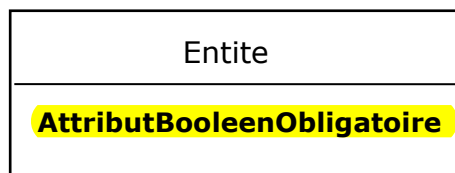
Notation sur le schéma entités-associations d'un attribut **facultatif** :



Un attribut **booléen** est un attribut dont la valeur est soit la valeur **vrai** soit la valeur **faux**.

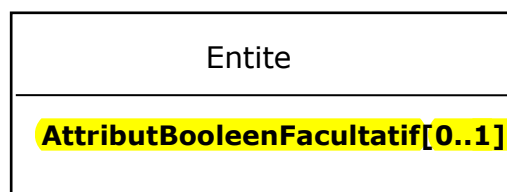
Un attribut booléen qui ne peut prendre que la valeur vraie ou fausse est donc un attribut **obligatoire** (soit vrai soit faux) ; les cardinalités sont [1..1]. Par conséquent, on ne note aucune cardinalité sur le schéma entités-associations pour des attributs booléens obligatoires.

Notation sur le schéma entités-associations d'un attribut **booléen obligatoire** :



Un attribut **booléen** peut cependant être **facultatif**, si les valeurs possibles sont **vrai**, **faux** et **inconnu** (null).

Notation sur le schéma entités-associations d'un attribut **booléen facultatif** :



Illustrons le choix d'attributs facultatifs et booléens sur base d'exemples tirés de la gestion d'un hôtel.

Exemple 1 : Le touriste peut payer par carte visa

Le touriste peut payer par carte visa ou par un autre moyen.

Les valeurs possibles pour l'attribut *Paie mentCarteVisa* sont : *vrai* ou *faux*.

Il s'agit donc d'un attribut **booléen obligatoire** (cardinalités [1..1] implicites).

Touriste
Paie mentCarteVisa

Exemple 2 : Le touriste peut avoir réservé son séjour à l'hôtel via une agence de voyage

Soit le touriste a réservé son séjour via une agence de voyage, soit il l'a réservé de sa propre initiative sans passer par une agence de voyage.

Les valeurs possibles pour l'attribut *ParAgence* sont : *vrai* ou *faux*.

Il s'agit donc d'un attribut **booléen obligatoire** (cardinalités [1..1] implicites).

Touriste
Paie mentCarteVisa ParAgence

Exemple 3 : On ne sait pas toujours si le touriste a réservé via agence de voyage ou non

Il se peut que, pour certains touristes, on ne sache pas s'ils sont passés par une agence de voyage.

Les valeurs possibles pour l'attribut *ParAgence* sont : *vrai*, *faux* ou *null* (valeur inconnue).

Il s'agit donc d'un attribut **booléen facultatif** (cardinalités [0..1]).

Touriste
Paie mentCarteVisa ParAgence[0..1]

Exemple 4 : Un touriste peut venir avec son propre véhicule

Les touristes peuvent arriver à l'hôtel avec leur propre véhicule ou via un autre moyen de transport. Si l'on veut mémoriser si le touriste est arrivé avec son véhicule personnel ou non, il s'agit d'un attribut **booléen obligatoire** (cardinalités [1..1] implicites).

Touriste
PaiementCarteVisa ParAgence[0..1] VehiculePersonnel

Exemple 5 : Si le touriste est venu avec son propre véhicule, le numéro de plaque de la voiture est mémorisé

En vue de contrôler les véhicules garés dans le parking de l'hôtel, on désire connaître les numéros de plaque des véhicules appartenant aux touristes logeant à l'hôtel. L'attribut *VehiculePersonnel* est transformé : il n'est plus de type booléen mais de type chaîne de caractères facultatif afin de mémoriser la plaque minéralogique éventuelle. Les touristes qui ont choisi une formule "voyage organisé en car", par exemple, ne communiqueront aucune plaque minéralogique : valeur inconnue pour l'attribut *VehiculePersonnel* (valeur *null*).

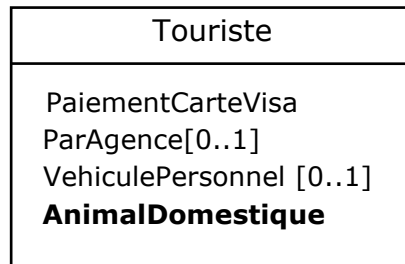
Il s'agit donc d'un attribut **facultatif** (cardinalités de l'attribut : [0..1]).

Touriste
PaiementCarteVisa ParAgence[0..1] VehiculePersonnel [0..1]

Exemple 6 : Le touriste peut être accompagné d'un animal domestique

Soit le touriste est accompagné, soit il n'est pas accompagné d'un animal domestique. Les valeurs possibles pour l'attribut *AnimalDomestique* sont : *vrai* ou *faux*.

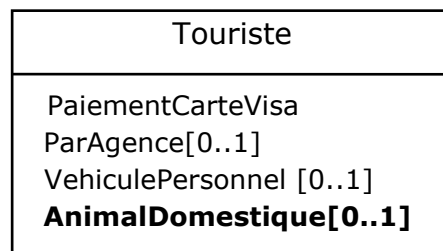
Il s'agit donc d'un attribut **booléen obligatoire** (cardinalités [1..1] implicites).



Exemple 7 : Si le touriste est accompagné d'un animal domestique, on mémorise de quelle espèce il s'agit

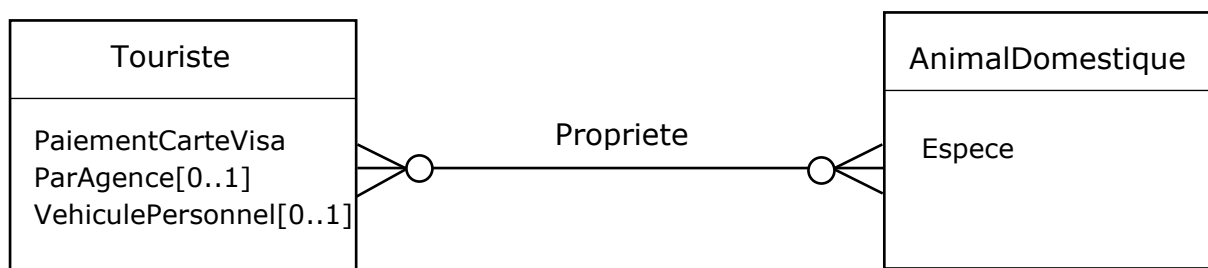
L'attribut *AnimalDomestique* est transformé : il n'est plus de type booléen mais de type chaîne de caractères facultatif afin de mémoriser l'espèce de l'animal domestique éventuel du touriste. Les touristes qui voyagent sans animaux domestiques auront la valeur *null* pour cet attribut.

Il s'agit donc d'un attribut **facultatif** (cardinalités de l'attribut : [0..1]).



Exemple 8 : Le touriste peut être accompagné d'animaux domestiques. Si le touriste est accompagné d'animaux domestiques, on mémorise la liste des espèces accompagnant le touriste

L'attribut *AnimalDomestique* est supprimé. On prévoit l'entité *AnimalDomestique* et on établit une association *Propriete* entre les entités *Touriste* et *AnimalDomestique*.



2.7 Non redondance dans le schéma entités-associations

faire des attributs pour ce qui peut être calculé est redondant. Le but d'une B.D. est de stocker, et donc de représenter, **toutes les données nécessaires** aux applications ultérieures. D'autre part, ne seront représentées que des données qui seront utilisées. Par conséquent, une B.D. doit être complète mais ne contiendra **pas de données inutiles** ni **redondantes**.

Une donnée est **redondante** si elle peut être **calculée ou dérivée** à partir de données déjà stockées dans la B.D.

💣 Une telle donnée ne sera **donc pas stockée sous forme d'attribut**, sauf introduction volontaire de redondance pour des raisons de performance.

Cette distinction à faire entre des attributs nécessaires et dérivables est essentielle, en particulier dans les bases de données dont seront extraites des statistiques.

☹ **Il s'agit d'une source fréquente d'erreurs auprès des étudiants !**

Exemple 1 : date de naissance et âge

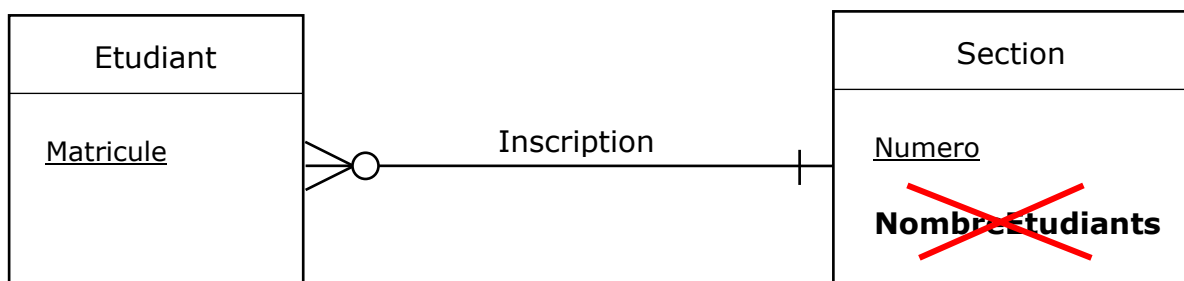
pas d'âge car calculable

Si ces deux données sont utilisées dans l'application, il suffit de prévoir l'attribut *DateNaissance* seul et en aucun cas un second attribut *Age* ; celui-ci peut être **calculé** par l'application à partir de l'attribut *DateNaissance*.

N.B. Il vaut mieux stocker la date de naissance plutôt que l'âge. En effet, le premier attribut a une valeur fixe, tandis que le second devrait régulièrement être mis à jour.

nb d'étudiants calculable par incrémentation

Exemple 2 : soit le schéma suivant

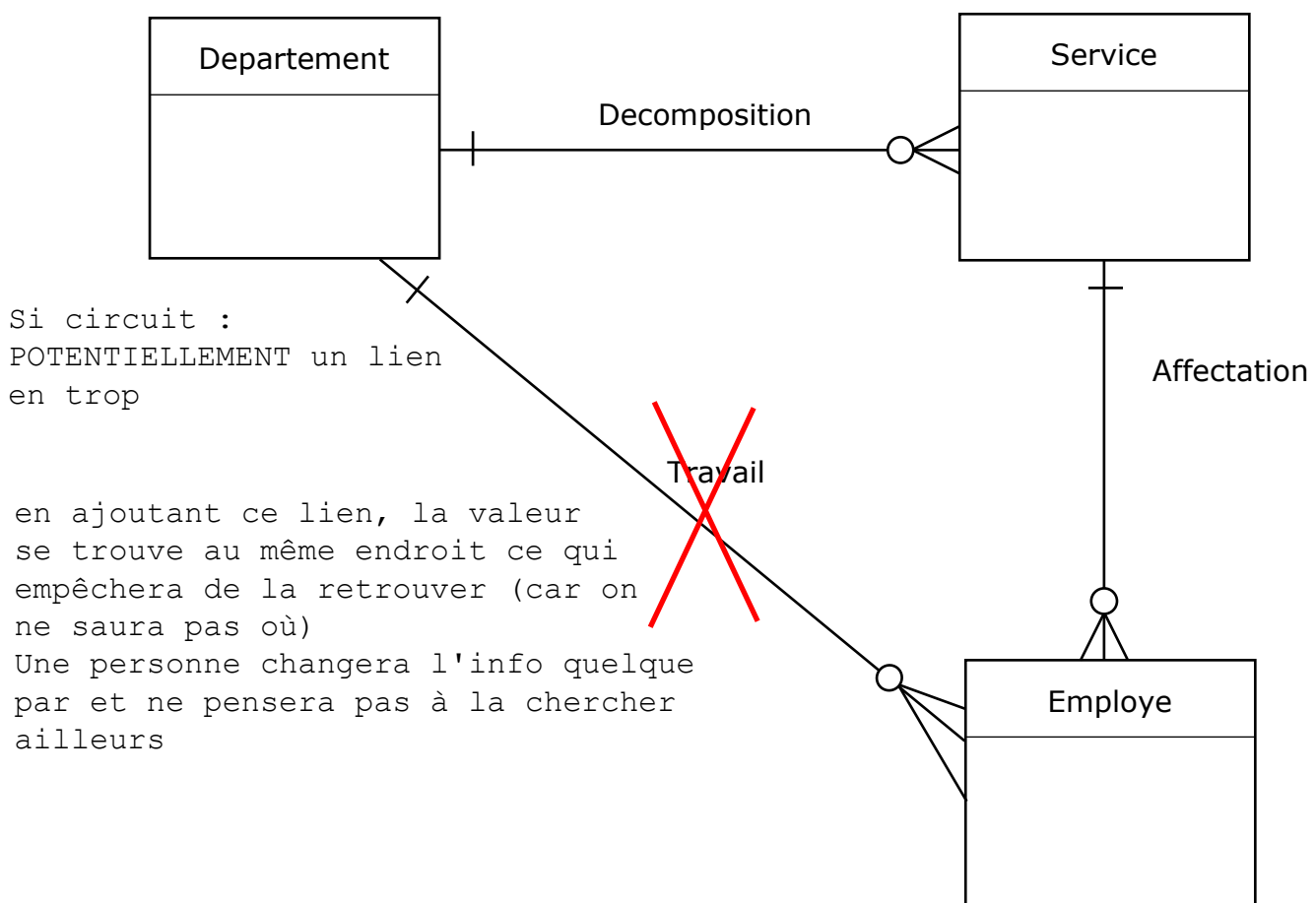


Si le nombre d'étudiants inscrits dans une section donnée doit être connu, il est **inutile** et théoriquement **faux** d'ajouter un attribut *NombreEtudiants* à l'entité *Section*. Cette donnée est **dérivable** à partir de l'association *Inscription*. En effet, l'existence de l'association *Inscription* signifie qu'il est possible, à partir d'un étudiant, de trouver sa section, et à partir d'une section de retrouver **tous** les étudiants qui y sont inscrits.

Il est possible de **demander au S.G.B.D. de compter ces liens** pour une section donnée.

Le principe est de **ne pas surcharger inutilement une B.D.**, sauf recherche de performance.

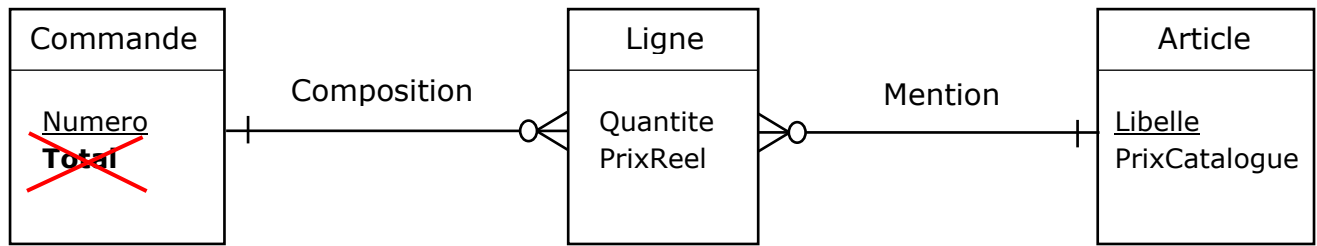
Exemple 3 : soit le schéma suivant



Grâce aux cardinalités 1-1, on sait qu'un employé est affecté à un seul service, et qu'un service appartient à un seul département. Par conséquent, un employé ne peut appartenir qu'à un seul département. Il est donc **inutile** et conceptuellement **faux** d'ajouter une association *Travail* entre les entités *Departement* et *Employe* ! Il s'agirait de **redondance**.

prixReel = prix actuel
Donc attribut utile même s'il peut désigner 2
choses identiques mais PAS de redondance

Exemple 4 : soit le schéma suivant



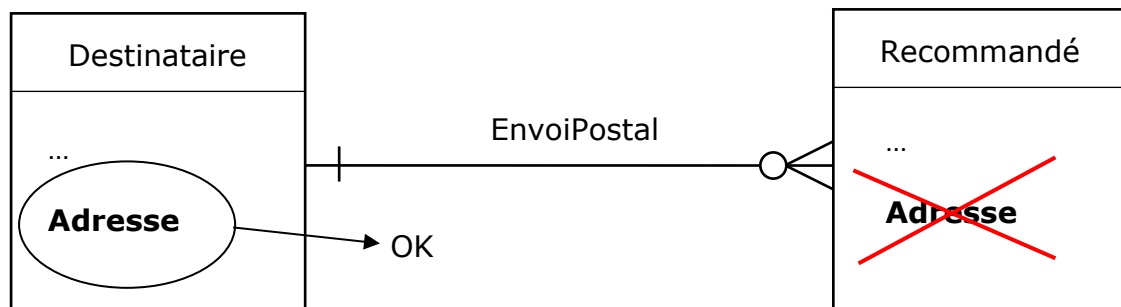
Une ligne (de commande) appartient à une seule commande et fait référence à un seul produit.

De plus, l'entité *Ligne* contient la quantité commandée (*Quantite*) et le prix réel accordé pour ce produit dans cette commande (*PrixReel*). Il est donc redondant d'ajouter un attribut mémorisant le total de la commande (*Total*) à l'entité *Commande*. C'est une donnée dérivable, calculable.

A noter que le prix réel d'une ligne de commande peut être différent du prix catalogue (par exemple, réduction accordée exceptionnellement au client, prix catalogue qui change après l'enregistrement de la ligne de commande dans la base de données...). Ces deux prix ne sont donc pas redondants.

Exemple 5 : Tous les recommandés doivent être envoyés au domicile du destinataire

Soit le schéma suivant



Où mémoriser l'adresse de destination des recommandés ?

Faut-il ajouter un attribut *Adresse* à l'entité *Destinataire* ou à l'entité *Recommandé* ?

Comme il faut viser le moins de redondance possible, il faut choisir la solution qui en offrira le moins. Puisque tous les recommandés d'un même destinataire doivent être envoyés à la même adresse, celle du domicile, un attribut *Adresse*

doit être attaché à l'entité *Destinataire* ; elle ne sera ainsi stockée qu'une seule fois dans la B.D. En effet, si l'on attache l'attribut *Adresse* à l'entité *Recommandé*, il y aura répétition de l'adresse du domicile du destinataire à chaque recommandé.

2.8 Pertinence des propositions

2.8.1 Contradiction des propositions

Une source de contradiction est l'évaluation des nombres d'éléments en liaison dans les propositions du type " A verbe B ".

Exemples :

- *Un véhicule a un propriétaire*
- *Un véhicule peut avoir plusieurs propriétaires.*

Deux réactions sont possibles (après discussion avec le client !) :

- ① Rejet des deux propositions contradictoires
- ② Choix de la **proposition la plus générale** :

"**un seul** propriétaire" est un cas particulier de "**plusieurs** propriétaires"

↳ On retient "*Un véhicule peut avoir **plusieurs** propriétaires*"

2.8.2 "Bruit" dans les propositions

bruit = quelque chose d'impertinent
dans la consigne

Il est possible que certaines propositions de l'énoncé ne doivent pas être prises en compte. Il s'agit de "**bruit**".

Exemple 1 : Les bulletins des étudiants seront générés 3 fois par an.

Cette information concerne la fréquence d'un **traitement**. Il ne faut donc pas prévoir de donnée supplémentaire dans la B.D.

Mais de telles propositions peuvent cacher une structure à incorporer dans la base de données.

Exemple 2 : Un jour de congé supplémentaire est accordé par tranche de 5 ans d'ancienneté.

A première vue, cette proposition ne concerne pas la base de données, mais le traitement qui calcule le nombre de jours de congé. Cependant, pour que le traitement puisse se faire, il faut connaître l'**ancienneté** de l'employé.

Il faut donc que l'ancienneté de chaque employé soit mémorisée dans la B.D.

Remarquons que puisque l'ancienneté évolue, il est préférable de stocker **la date d'entrée en fonction** qui, elle, reste fixe.

Exemple 3 : Le montant en devise de la facture sera calculé à partir du taux de la devise atteint le jour de la commande

Cela ressemble à du bruit : il s'agit d'une information sur la façon de calculer le montant de la facture. Mais pour que le traitement puisse se faire, il faut mémoriser **la date de la commande** afin de retrouver le cours de la devise ce jour-là, ou carrément le **taux de change**.

Exemple 4 : Les étudiants que la Haute Ecole forme ...

Supposons que la base de données ne concerne qu'une seule et même Haute Ecole. Il est donc inutile de prévoir l'entité *HauteEcole* ; elle n'aurait qu'une seule occurrence.

En conclusion,

La base de données doit comprendre toutes les données nécessaires aux traitements, sans redondance inutile

2.8.3 Autres suggestions

Encore quelques conseils découlant du bon sens.

① Si un concept n'a qu'une caractéristique, c'est-à-dire **si une entité ne contient qu'un seul attribut**, il est peut-être nécessaire de la remettre en question ! Ce concept n'est **peut-être qu'un simple attribut** d'une autre entité.

Exemple : Supposons qu'on ne mentionne que le nom de la catégorie des produits.

Version 1 :



Puisque d'après les cardinalités 1-1, un produit n'est que d'un seul type, et qu'on ne mentionne que le nom de la catégorie (qui est identifiant), on pourrait en faire un simple attribut de l'entité produit.

Version 2 :

Produit
<u>Nom</u> Libelle NomCategorie

A noter cependant qu'il y a une utilité à garder l'entité *Categorie* : en effet, la présence de l'entité *Categorie* permet de définir le catalogue des catégories existantes, qui plus est, est évolutif (on peut ajouter de futures catégories par la suite dans la B.D.) et d'assurer qu'on relie toujours un produit à une catégorie existante. La simple présence de l'attribut *NomCategorie* dans l'entité *Produit* permet d'introduire des valeurs de catégories non existantes ou mal orthographiées.

② Si **une même caractéristique** (d'un concept) est citée à **plusieurs endroits**, il faut peut-être en faire une **entité**.

Exemple : Le nom du fournisseur apparaît à plusieurs endroits

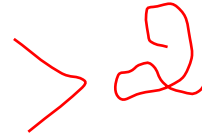
On peut envisager la création d'une entité *Fournisseur* qui contiendra (au minimum) l'attribut *Nom* et la création d'une association pour relier cette entité *Fournisseur* aux concepts le mentionnant auparavant.

Cela permet, entre autres, de pouvoir ajouter dans le futur des caractéristiques supplémentaires aux fournisseurs sans induire de redondance, ou encore de prévoir le catalogue des fournisseurs. Ce dernier est dynamique ; il peut évoluer au fil du temps (on peut ajouter de futurs fournisseurs dans la B.D.).

Il faut toujours peser le pour et le contre (avantages et inconvénients) quand on choisit d'augmenter le nombre d'entités.

2.9 Associations particulières

2.9.1 Relation de degré supérieur à 2



💣 Attention, il y a des propositions complexes qu'il ne faut pas réduire à des propositions élémentaires de type "A verbe B", sous peine de perdre des informations !

Exemple : Un **client** achète un **produit** chez un **fournisseur**.

N.B. On souhaite mémoriser dans la base de données les **habitudes, les préférences** d'achat, c'est-à-dire le fournisseur chez lequel un client est allé s'approvisionner au moins une fois pour un produit donné.

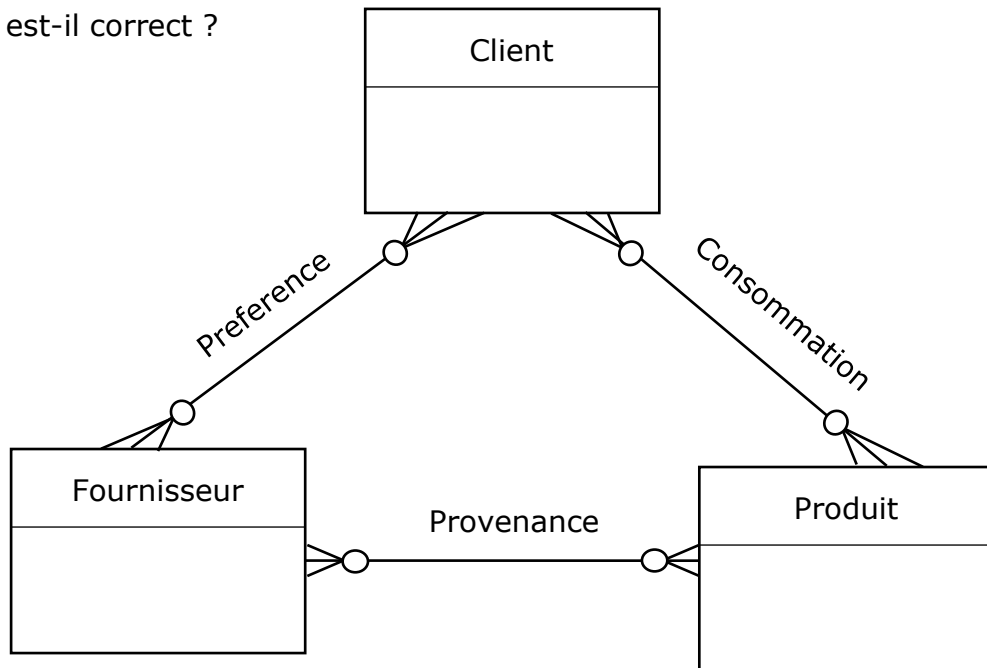
Par exemple, *Luc a acheté un PC chez Priminfo.*

Si Luc a acheté trois PC chez Priminfo, cela ne fait l'objet que d'une occurrence d'achat dans la base de données.

Il faut donc pouvoir stocker des occurrences du type :

- **Luc** achète un **PC** chez **Priminfo**
- **Paul** achète une **imprimante** chez **Flag 2000**
- **Anne** achète du **papier** chez **Dell**
- **Luc** achète du **papier** chez **Flag 2000**
- **Laure** achète une **imprimante** chez **Dell**
- **Anne** achète un **scanner** chez **Priminfo**
- **Paul** achète un **PC** chez **Flag 2000**
- **Louis** achète un **scanner** chez **Dell**
- **Laure** achète du **papier** chez **Flag 2000**

Ce schéma est-il correct ?



Un tel schéma permet de retrouver :

- Les fournisseurs d'un client :
Exemple : Priminfo et Flag 2000 pour Luc
- Les fournisseurs d'un produit :
Exemple : Priminfo et Flag 2000 pour les PC
- Les produits consommés par un client :
Exemple : PC et papier pour Luc

Il y a malheureusement **perte d'informations** : il est, par exemple, impossible de retrouver *chez qui Luc a acheté un PC* ou *à quel(s) client(s) Flag 2000 vend du papier*. Ce schéma est donc **faux**.

Il y a un concept nouveau qui représente la proposition complexe toute entière ; c'est le concept **achat**.

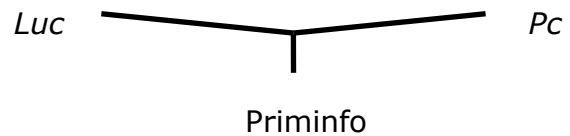
Une telle proposition doit donc être reformulée comme suit :

- ① Les clients effectuent des **achats**
- ② Tout **achat** concerne un produit
- ③ Les **achats** s'effectuent chez les fournisseurs

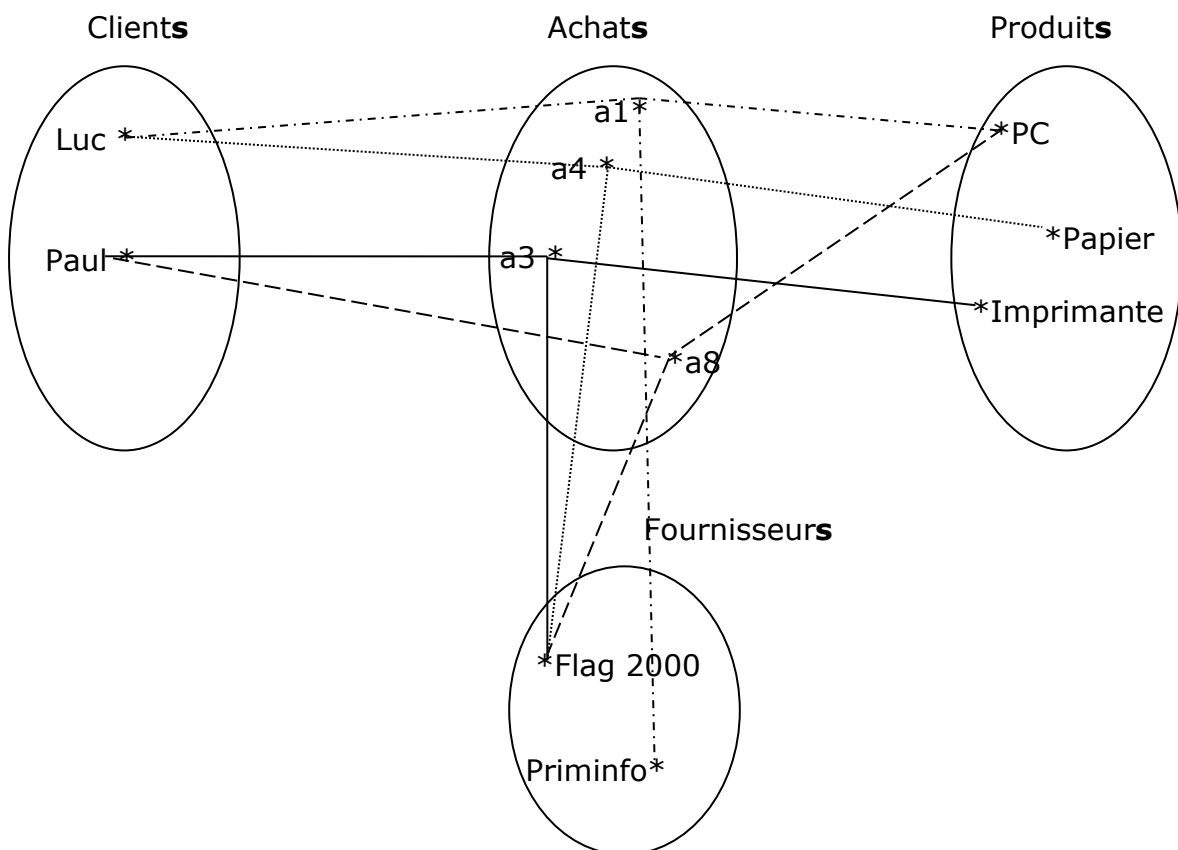
Un achat est un **lien**, une relation **entre** des **clients**, des **produits** et des **fournisseurs**. Il s'agit d'une relation **ternaire** (c'est-à-dire de degré 3).

Toute **occurrence** d'achat est un lien entre une occurrence de *Client*, une occurrence de *Produit* et une occurrence de *Fournisseur*. Toute occurrence d'*Achat* est donc une "**étoile**" à exactement 3 branches.

Exemple d'occurrence d'Achat :



Au point de vue des **ensembles d'occurrences** :



Le concept d'achat est représenté par une **nouvelle entité Achat** reliée aux trois entités *Client*, *Produit* et *Fournisseur* via 3 nouvelles associations.

Calcul des cardinalités des 3 associations :

- Au point de vue *Achat* :

Une occurrence d'achat est liée à exactement une et une seule occurrence de *Client*, une et une seule occurrence de *Produit* et une et une seule occurrence de *Fournisseur*.

- Au point de vue *Client* :

A combien d'occurrences d'*Achat* est reliée au minimum et au maximum une occurrence de *Client* ? Minimum = 0 et Maximum = plusieurs.

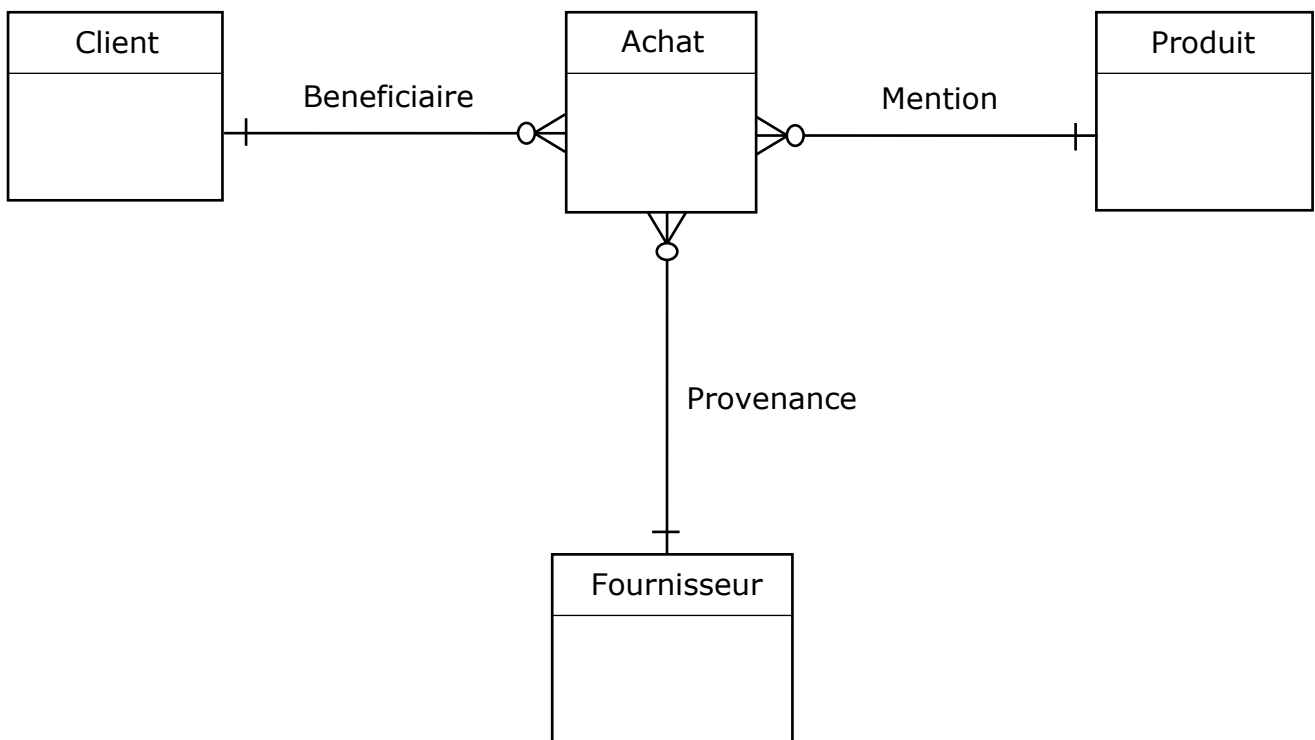
- Au point de vue *Produit* :

A combien d'occurrences d'*Achat* est reliée au minimum et au maximum une occurrence de *Produit* ? Minimum = 0 et Maximum = plusieurs.

- Au point de vue *Fournisseur* :

A combien d'occurrences d'*Achat* est reliée au minimum et au maximum une occurrence de *Fournisseur* ? Minimum = 0 et Maximum = plusieurs.

Le schéma d'une relation ternaire est



A noter qu'une entité ne représente une **vraie relation ternaire** entre 3 autres entités que si les 3 associations sont de type **1 à plusieurs**.

Par conséquent, si une de ces associations est de type 1 à 1, c'est qu'il ne s'agit pas d'une vraie relation ternaire. Cette "pseudo" relation ternaire peut alors être réduite à de simples relations binaires.

Ce principe peut être généralisé à toute relation de degré > 2 (quaternaire, ...).

Exemple :

Dans la gestion des réservations de place sur des vols dans une compagnie aérienne, un couple pilote/operateur-radio est associé à chaque vol. Les couples pilote/opérateur-radio ne sont pas fixes, ils varient d'un vol à l'autre.

Cette formulation pourrait laisser sous-entendre qu'il existe une relation ternaire *Pilotage* entre les entités *Vol*, *Pilote* et *OperateurRadio*. Or, comme un vol n'est associé qu'à un couple pilote/operateur-radio, il y aurait une association 1 à 1 entre les entités *Vol* et *Pilotage*. Il ne s'agit donc pas d'une vraie relation ternaire.

Schéma incorrect

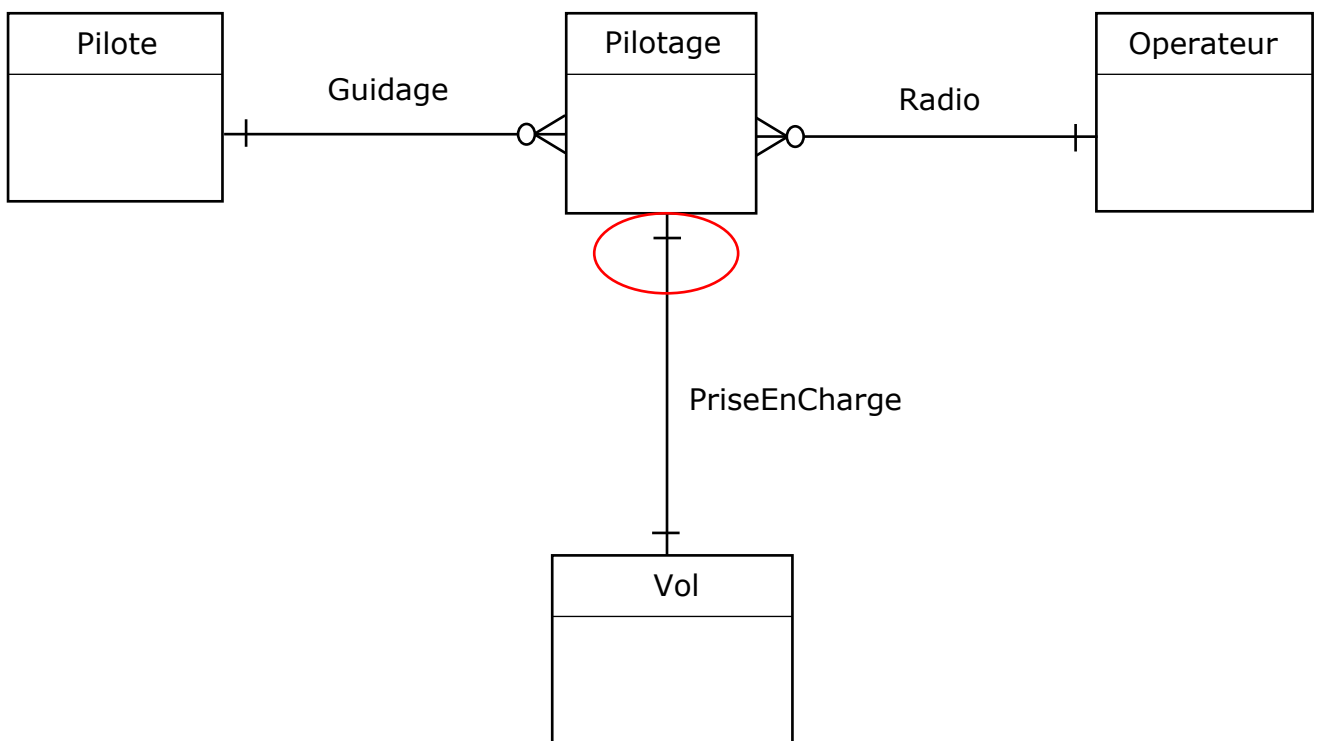
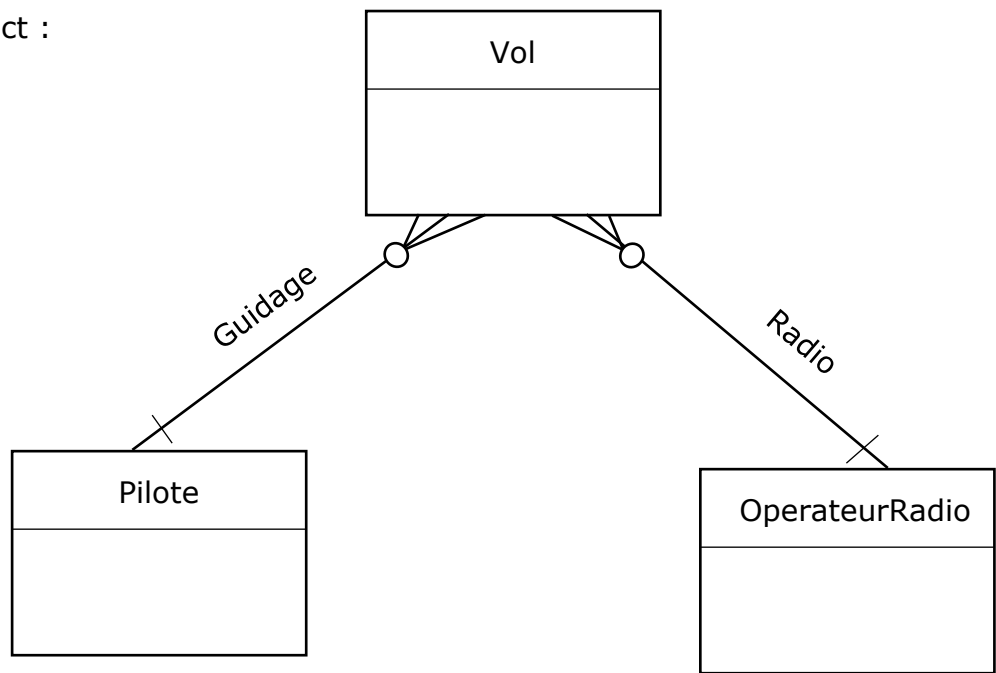


Schéma correct :



2.9.2 Association cyclique

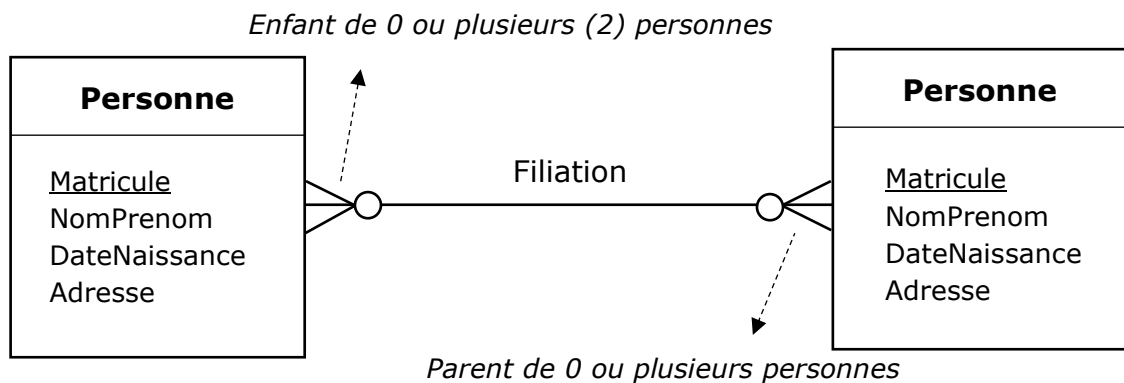
Les associations cycliques sont aussi appelées associations **réflexives** ou associations **récurrentes**.

Exemple 1

Supposons qu'on veuille représenter des liens de filiation (liens parents-enfants) entre individus.

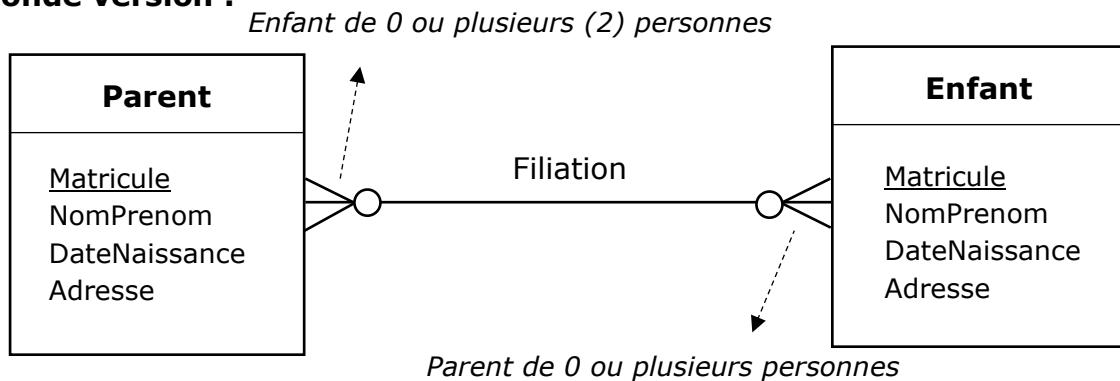
Les schémas ci-dessous sont-ils corrects ?

Première version :



Deux entités qui apparaissent sur un schéma entité-association sont supposées représenter deux concepts distincts. Ce schéma est donc **incorrect**.

Seconde version :

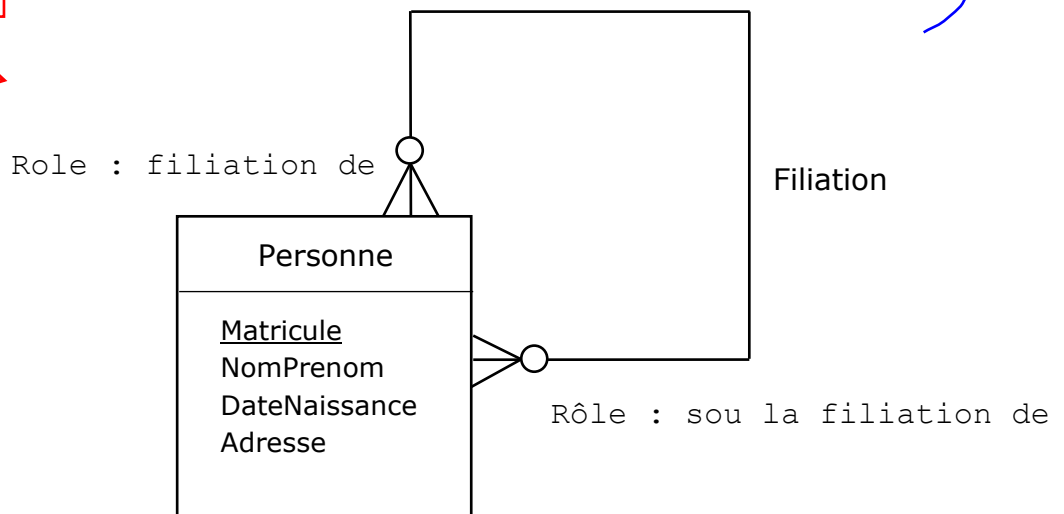


Tout enfant peut lui-même être parent. Si c'est le cas, les informations concernant la même personne seront représentées **deux fois** dans la B.D. D'où **redondance**. Ce schéma est également à rejeter.

Il est donc nécessaire de pouvoir représenter des associations dites **cycliques**.

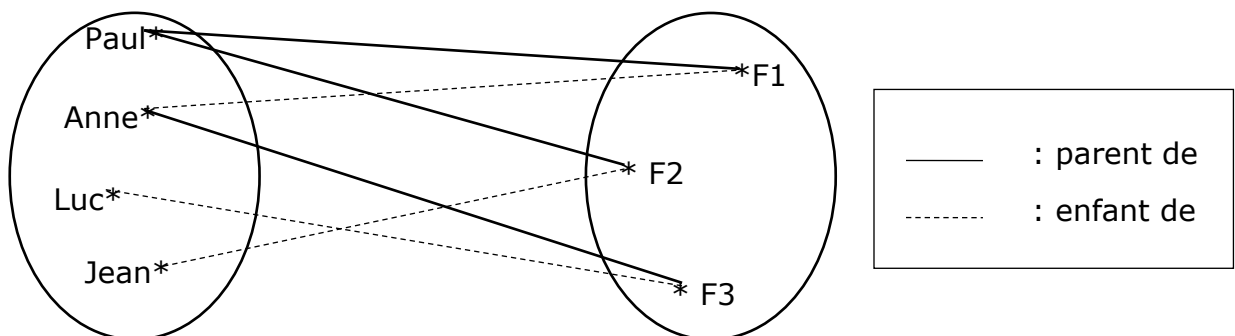
Le schéma suivant est **correct** et **dépourvu de toute redondance** :

asso cyclique

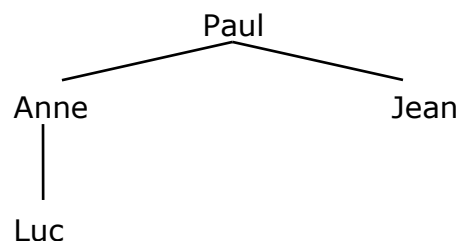


N.B. L'association cyclique *Filiation* ne signifie pas qu'une même occurrence de *Personne* a un lien de filiation avec elle-même. Il s'agit de liens entre des occurrences distinctes de *Personne*.

Au niveau occurrences :



Ces trois occurrences de *Filiation* (F1, F2 et F3) permettent de représenter la portion d'arbre généalogique suivante :



Si les liens entre conjoints doivent aussi être représentés, le schéma devient :



Dans les associations cycliques 1 à N, il y a une sorte d'ambiguïté. En effet, la même entité joue deux rôles différents ; il est donc important de savoir à quel rôle correspondent quelles cardinalités, surtout si celles-ci sont différentes pour les deux rôles. Un schéma contenant des associations cycliques de type 1 à N devrait être accompagné d'une documentation précisant à quel rôle est associée quelle cardinalité.



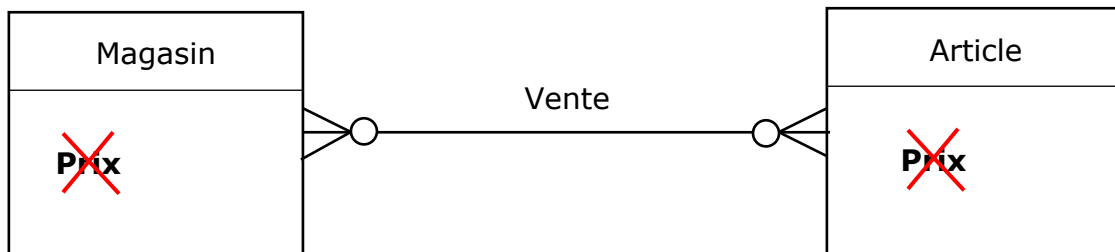
2.9.3 Relation avec attributs



Dans le cas où on a identifié une relation plusieurs à plusieurs, une caractéristique dont la valeur dépend à la fois des deux entités reliées est une **caractéristique de la relation**. Dans ce cas, la relation plusieurs à plusieurs sera représentée par une nouvelle entité et cette caractéristique sera un attribut de la nouvelle entité.

Exemple 1 :

Un magasin vend des articles et un article peut être vendu par plusieurs magasins. Le prix de vente d'un article dépend du magasin

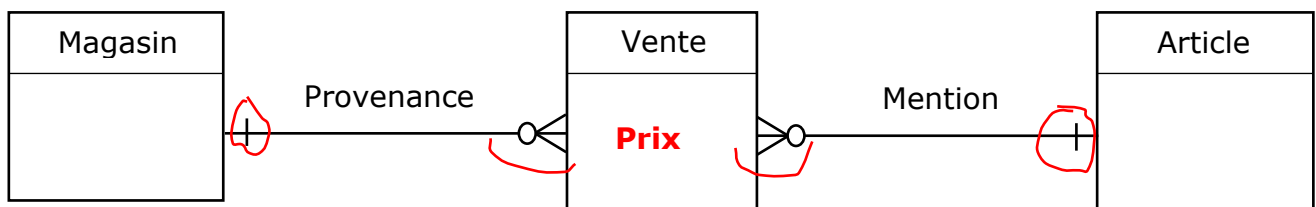


L'attribut *Prix* ne peut pas être associé à *Magasin* car un magasin n'a pas de prix fixe pour tous ses articles.

L'attribut *Prix* ne peut pas être associé à *Article* car un article n'est pas vendu au même prix dans tous les magasins.

Le prix varie en fonction de l'article ET du magasin. Le prix est donc bien une caractéristique du lien *Vente*.

La relation plusieurs à plusieurs *Vente* est représentée dans ce cas par une nouvelle entité. Le *prix* est un attribut de cette entité *Vente*.

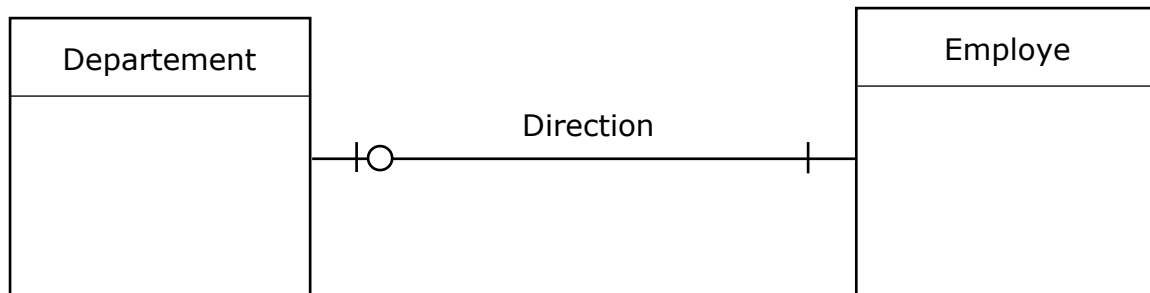


association de vient une entitée

Si N à N et si une caractérisitique dépend à la fois de 2 identitée
F. Dubisy

Exemple 2 :

Reprenons l'association *Direction* entre les entités *Departement* et *Employe* : un employé ne peut être directeur que d'un département à la fois et un département doit avoir un et un seul directeur.



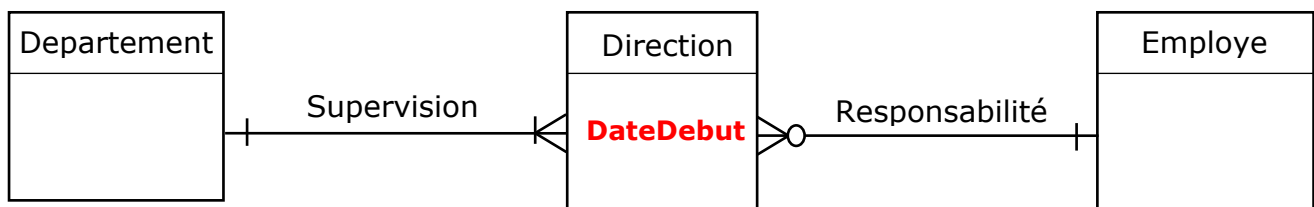
Dans ce cas, on ne considère que les directeurs **courants** ; on ne mémorise pas l'historique des directeurs des départements.

Que faut-il modifier dans le schéma ci-dessus pour tenir compte de cet historique ?

Les cardinalités maximales 1 passent à N. En effet, un même département peut avoir vu se succéder plusieurs directeurs. D'autre part, un même employé peut avoir dirigé plusieurs départements à des moments différents dans le temps.

D'autre part, si l'on désire pouvoir retrouver le directeur **actuel** de chaque département ainsi que **l'ordre de succession des différents directeurs** d'un même département, il faut ajouter l'attribut *DateDebut* au lien *Direction* spécifiant la date de début de chaque mandat de directeur.

L'association *Direction* est représentée dans ce cas par une nouvelle entité contenant l'attribut *DateDebut*.



Pour retrouver le directeur actuel d'un département donné, il suffit de retrouver, parmi toutes les occurrences de l'association *Direction* concernant ce département, celle qui a la date de début la plus récente.

Si l'on désire retrouver tous les directeurs d'un département donné et les classer par ordre de succession au poste de directeur, il suffit de retrouver toutes les occurrences de *Direction* concernant ce département et de les trier par date de début croissante.

Notons qu'il est inutile d'associer des attributs à des associations

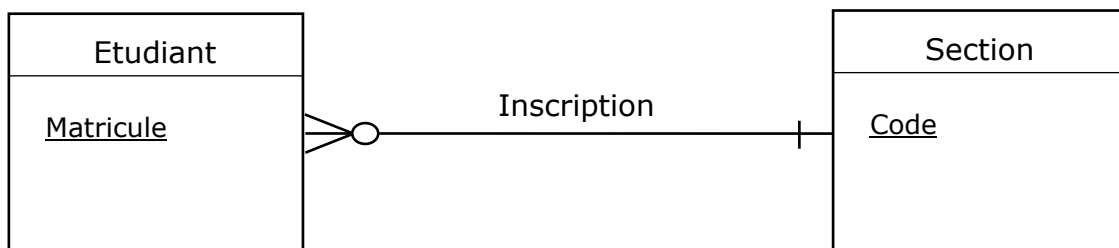
1 à N.

Cas \neq

5

En effet, l'attribut est alors une caractéristique d'une des deux entités.

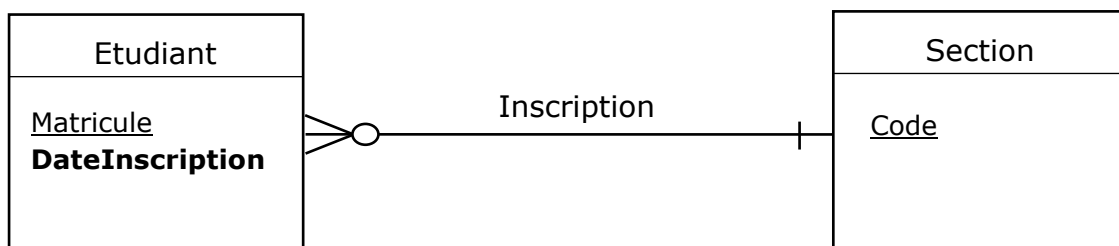
attribut en plus dans un énoncé ne veut pas dire qu'on doit rajouter un attribut
Exemple 1 : \Rightarrow Cardinalité



Quid de la date d'inscription ?

La date d'inscription pourrait être considérée d'un point de vue logique comme une caractéristique de l'association *Inscription*.

Cependant, comme un étudiant n'est relié qu'à un seul lien inscription, la date d'inscription peut être placée dans l'entité *Etudiant* ; la date d'inscription devenant alors une caractéristique de tout étudiant.

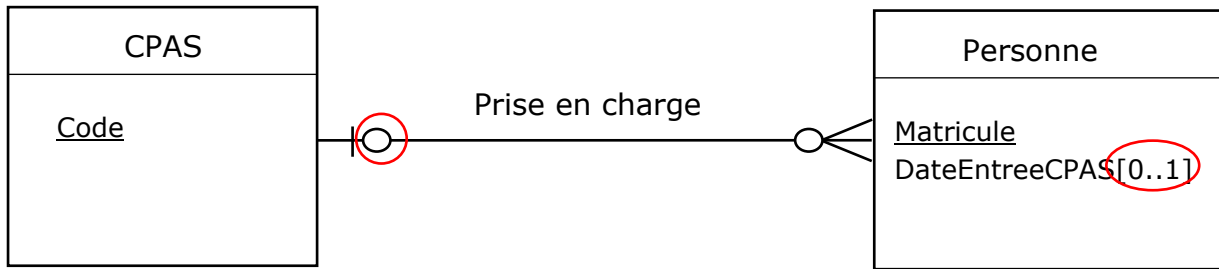


Dans le cas d'attribut d'une association 1 à N :

\Rightarrow Placer l'attribut dans l'entité dont chaque occurrence ne participe qu'à une seule occurrence de l'association

Si l'association est facultative pour cette entité, l'attribut placé dans l'entité est alors **facultatif**.

Exemple 2 :



2.10 Contraintes additionnelles

Le **formalisme** du schéma entité-association est **limité** ; il ne permet pas de représenter toutes les informations. Il ne suffit donc pas à lui seul. **Une documentation annexe complètera le schéma.**

Cette documentation comprendra :

- ① La définition de tous les termes apparaissant sur le schéma :
 - Entités
 - Associations
 - Attributs
- ② Des **contraintes** dites **additionnelles** car complétant le schéma.

Ces contraintes peuvent être exprimées en langage naturel (français, anglais...) ou en pseudo-langage (cf exemples ci-dessous).

Contraintes d'identifiant

Une entité peut avoir **un ou plusieurs** identifiant(s). Il peut également n'en avoir **aucun**, même si cela n'est pas conseillé.

☀ Rappelons que, au cours de son existence, une occurrence d'entité ne peut voir la valeur de son identifiant changer. Un identifiant doit rester **fixe**.

2.10.1 Identifiant composé d'attributs

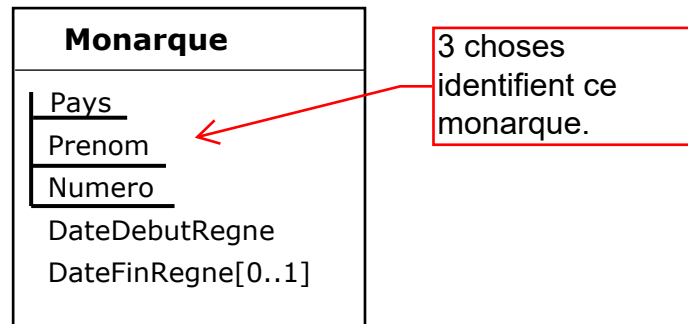
① Le cas le plus simple est celui d'un identifiant unique **composé d'un seul attribut**. Ce dernier est alors souligné.

Exemple :

Ouvrage
<u>Numero</u>

② Une entité peut également posséder un identifiant **composé de plusieurs attributs**. Ils sont soulignés et reliés par une ligne verticale.

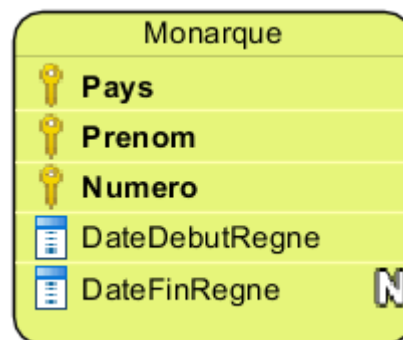
Exemple 1 :



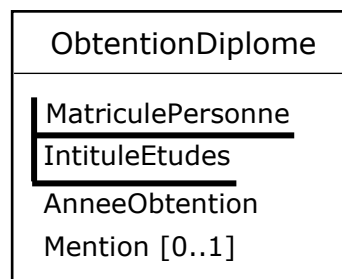
Citons à titre d'exemples les rois Henri IV de France, Henri IV d'Angleterre, Louis XIV de France, Philippe I^{er} de France, Philippe I^{er} de Belgique, Elisabeth II d'Angleterre...

Pour rappel, certains environnements ou outils désignent la clé primaire sous forme d'un symbole représentant une clé. Dans ces environnements, si un identifiant est composé, chaque attribut composant la clé primaire est précédé du symbole de la clé.

Exemple en Visual Paradigme :



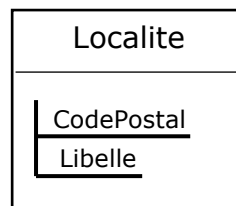
Exemple 2 :



Les intitulés des études sont, par exemple, Master en chimie, Doctorat en médecine, Baccalauréat en informatique de gestion... Une même personne peut avoir obtenu plusieurs diplômes et les mêmes études peuvent avoir été suivies

avec succès par plusieurs personnes avec ou sans mention (les mentions étant : Distinction, Grande distinction, Plus grande distinction et La plus grande distinction).

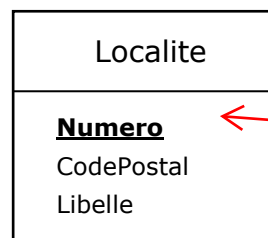
Exemple 3 :



Pour rappel, en Belgique, le code postal seul n'est pas identifiant (exemples : 5020 Daussoulx, 5020 Malonne, 5020 Vedrin). De même, le libellé seul n'est pas identifiant (exemples : 6230 Buzet dans la province du Hainaut et 5530 Buzet dans la province de Namur). La combinaison des deux attributs peut cependant être considérée comme un identifiant naturel.

N.B. **Les identifiants composés de plusieurs attributs sont rarement utilisés en entreprise** : on y privilégie en général un **identifiant technique** composé d'un seul attribut plutôt que d'utiliser un identifiant naturel composé de plusieurs attributs. En effet, en base de données relationnelles, la création des clés étrangères qui traduisent les types d'associations 1 à N en est simplifiée (cf. section 3.4.2. Représentation des associations en base de données relationnelles).

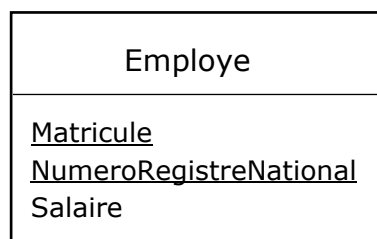
Exemple :



mais en général on met un identifiant technique (ici numéro)

③ Une entité peut posséder **plusieurs identifiants**. Chaque attribut identifiant ou groupe d'attributs identifiant est souligné, mais indépendamment des autres.

Exemple :

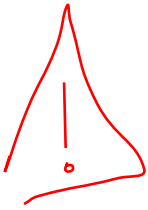


Un employé a 2 identifiants :

- Matricule
- NumeroRegistreNational

A noter que dans certains environnements tel que Visual Paradigm, il n'y a pas de formalisme permettant de visualiser plusieurs identifiants sur la même entité. Une seule clé primaire est permise, éventuellement composée de plusieurs attributs, chacun étant associé à un symbole de clé. Si l'on désire malgré tout préciser plusieurs identifiants pour une entité dans de tels environnements, il faudra alors avoir recours à des contraintes additionnelles à inscrire dans la documentation qui accompagnera le schéma.

2.10.2 Identifiant hybride



Est appelé identifiant **hybride** un identifiant composé de :

0, 1 ou plusieurs attribut(s)

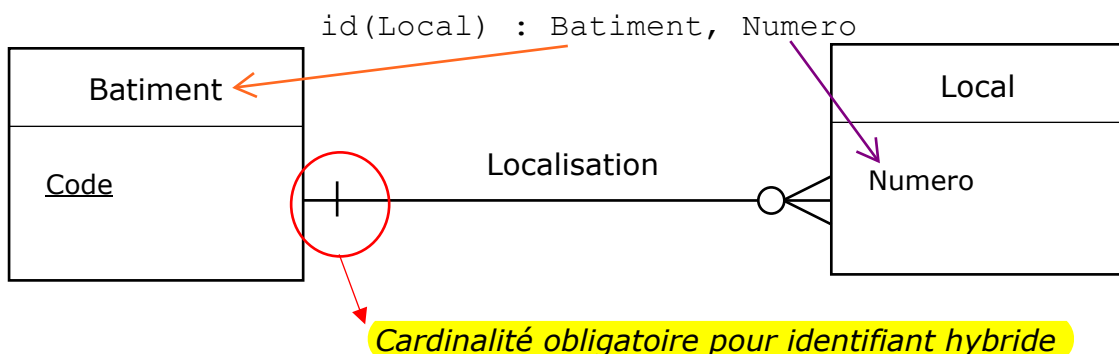
ET

un ou plusieurs rôle(s) via une ou plusieurs association(s).

En effet, l'identifiant d'une entité peut dépendre de l'identifiant d'une autre entité qui lui est reliée via une association.

Exemple 1 :

Tous les locaux d'un même bâtiment portent des numéros différents mais des locaux dans des bâtiments différents peuvent porter le même numéro. Par exemple, un local portant le numéro 12 existe à la fois dans le bâtiment X25 et dans le bâtiment K86, mais il n'y a pas deux locaux numéro 12 ni dans le bâtiment X25 ni dans le bâtiment K86.



La valeur de *Numero* pour un local particulier est unique au sein d'un même bâtiment. **L'attribut *Numero* est donc identifiant au sein de l'ensemble des locaux appartenant à un même bâtiment.**

L'attribut *Numero* de *Local* n'est pas identifiant au sein de la B.D. ; il n'est donc **pas souligné** sur le schéma. L'entité *Local* n'a pas d'identifiant constitué uniquement d'attributs.

Le formalisme des schémas entité-association ne permet pas de représenter ce type d'identifiant **hybride**. Il faut donc avoir recours à une **contrainte additionnelle** exprimée en pseudo-langage. L'identifiant de *Local* ($id(Local)$) est donc constitué de l'identifiant de *Batiment* (on se contentera de noter le nom de l'entité reliée, soit *Batiment* ici) et de l'attribut *Numero*.

L'identifiant de l'entité *Local* est spécifié via la contrainte additionnelle suivante :

id (Local) : Batiment, Numero

Attention, un identifiant hybride n'est possible **que s'il y a des cardinalités 1-1** sur le rôle intervenant dans l'expression de l'identifiant hybride. Ces cardinalités doivent impérativement être 1-1 et non 0-1, encore moins 0-N.

N.B. S'il existe plusieurs associations entre *Batiment* et *Local*, il est indispensable de préciser dans la contrainte via quelle association l'entité *Batiment* intervient dans l'identifiant de *Local*, soit l'association *Localisation* (**sinon risque d'ambiguïté**). La contrainte devient :

id (Local) : Localisation.Batiment, Numero

Exemple 2 :

*Dans une compagnie d'assurances, un client signe des contrats et un contrat n'est signé que par un seul client. Supposons que les **contrats signés par un même client** possèdent tous un numéro d'ordre (NumeroOrdre) **différent**.*

Un contrat n'a pas un numéro unique au sein de la compagnie ; l'entité *Contrat* n'a donc pas d'identifiant constitué uniquement d'attributs.

L'attribut *NumeroOrdre* est donc identifiant au sein de l'ensemble des contrats appartenant à un même client.



Cardinalité obligatoire pour identifiant hybride

L'identifiant de l'entité *Contrat* est spécifié via la contrainte additionnelle :

id(Contrat) : Client, NumeroOrdre

Ou, s'il y plusieurs associations entre les entités *Client* et *Contrat*, par la contrainte additionnelle suivante :

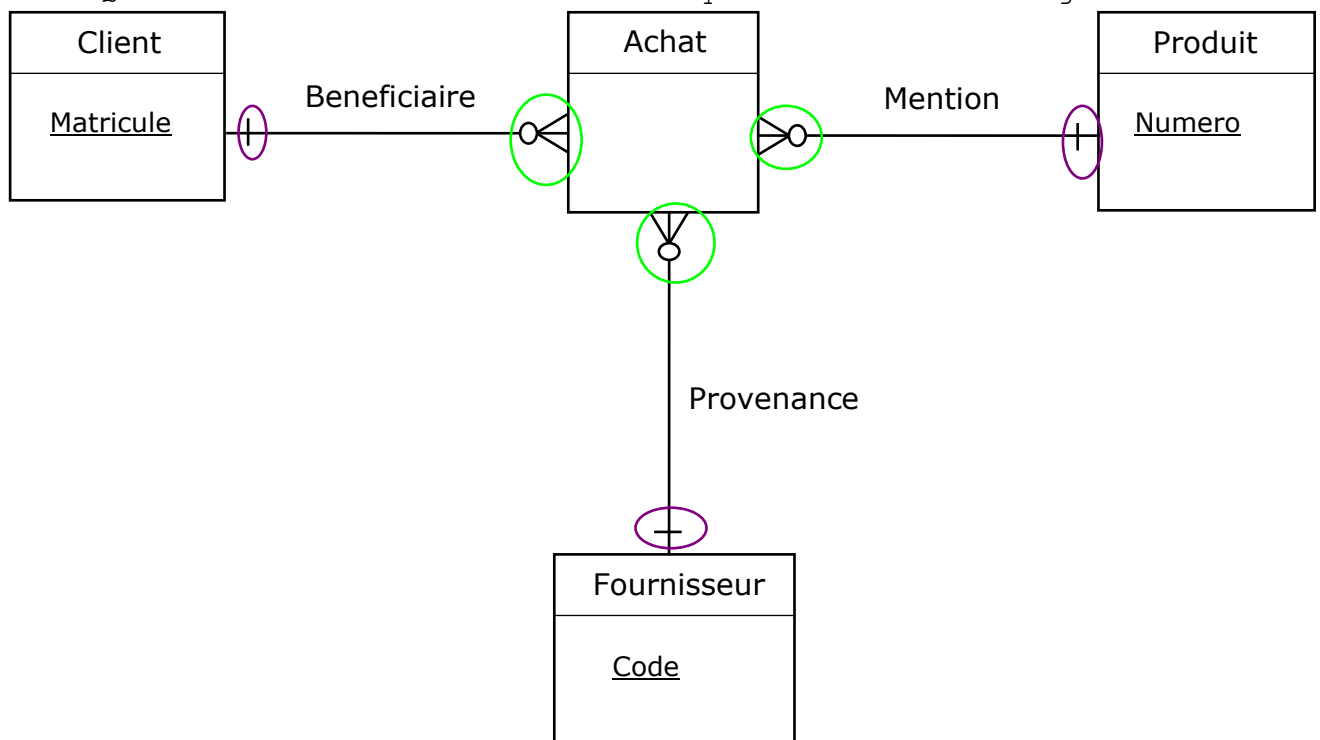
id(Contrat) : Signature.Client, NumeroOrdre

Exemple 3 :

Reprenons l'exemple des habitudes, des préférences d'achat, c'est-à-dire le fournisseur chez lequel un client est allé s'approvisionner au moins une fois pour un produit donné.

L'entité *Achat* représente une relation ternaire entre les entités *Client*, *Produit* et *Fournisseur*.

Quand on est dans une N à N et qu'on demande de rajouter un attribut



N.B. Pour rappel, dans la base de données correspondant à ce schéma, une occurrence d'achat mémorise les habitudes (préférences) d'achat, c'est-à-dire le fournisseur chez lequel un client est allé s'approvisionner (au moins une fois) pour un produit donné. Par exemple, *Luc* est allé s'approvisionner en *PC* chez

Priminfo. Si *Luc* a acheté trois *PC* chez *Priminfo* au cours du mois, cela ne fait l'objet que d'**une** occurrence d'achat dans la base de données.

Quel est l'identifiant de l'entité *Achat* ?

Toute occurrence d'*Achat* est un lien entre exactement **une occurrence de Client, une occurrence de Produit et une occurrence de Fournisseur**.

Chaque occurrence d'*Achat* est donc identifiée par la combinaison de l'identifiant du client, de l'identifiant du produit et de l'identifiant du fournisseur.

L'occurrence d'achat qui correspond au fait que *Luc* est allé un jour acheter des *PC* chez *Priminfo* est identifiée au sein de la base de données par la combinaison: *Luc + PC + Priminfo*.

L'identifiant de l'entité *Achat* est spécifié via la contrainte additionnelle :

id(Achat) : Client, Produit, Fournisseur

Exemple 4 :

Reprenons l'exemple ci-dessus. Supposons que ce qui nous intéresse ce n'est pas d'enregistrer chez qui quel client s'approvisionne et quel produit il y achète, mais bien les achats réellement effectués par les clients.

Dans ce cas, si *Luc* a acheté trois fois des *PC* chez *Priminfo* à des dates différentes, on mémorisera trois occurrences d'achats différentes dans la base de données, en précisant à chaque fois la date de l'achat ainsi que la quantité de produit achetée et le prix réel obtenu (occasionnellement, une réduction sur le prix catalogue pourrait avoir été octroyée après négociation).

Il faut alors ajouter trois attributs à l'entité *Achat*, à savoir les attributs *PrixUnitaire*, *Quantite* et *Date*.

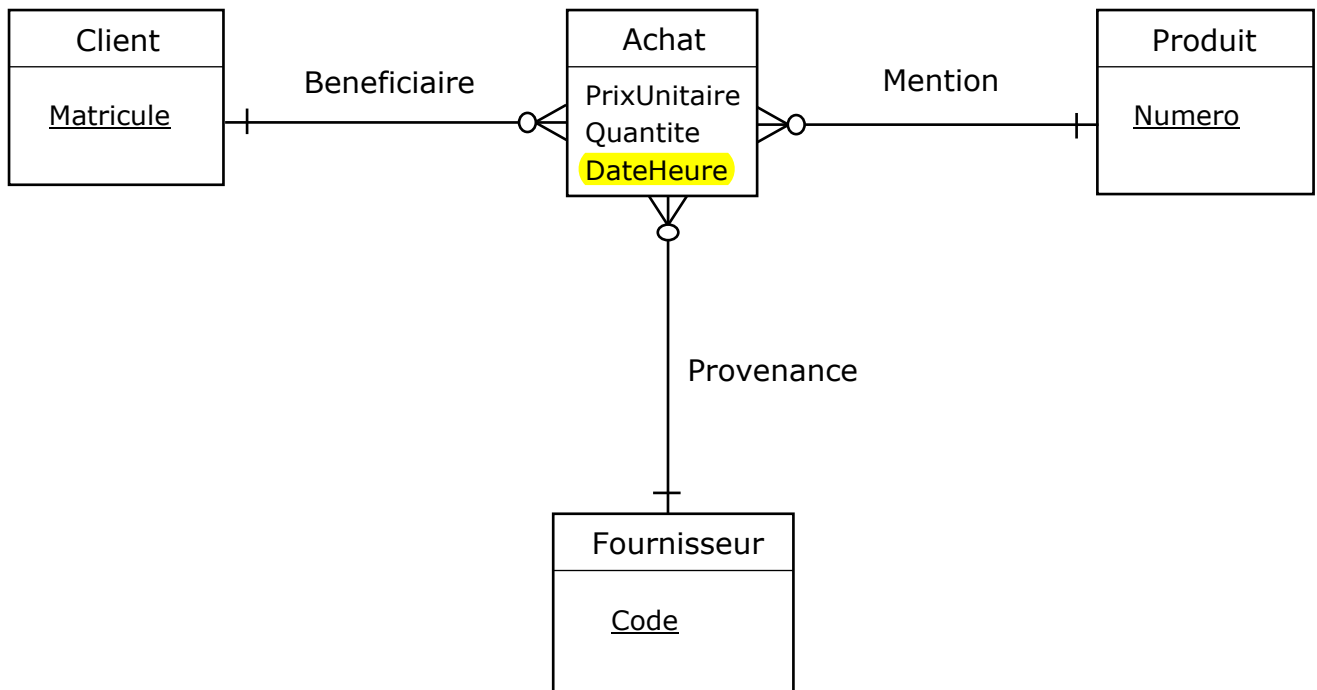
Chaque occurrence d'*Achat* est ici identifiée par la combinaison de l'identifiant du client, de l'identifiant du produit, de l'identifiant du fournisseur et de la date (+ heure) à laquelle l'achat a été effectué.

Ainsi par exemple, l'achat par *Luc* de 3 *PC* chez *Priminfo* le 22/2/2019 à 14h32 est identifié au sein de la base de données par la combinaison :

Luc + PC + Priminfo + 22/2/2019-14h32

L'achat par *Luc* de 2 *PC* chez *Priminfo* le 28/2/2019 à 10h35 est identifié au sein de la base de données par la combinaison :

Luc + PC + Priminfo + 28/2/2019-10h35



Il faut noter que ni le prix négocié, ni la quantité n'interviennent dans l'établissement de l'identifiant d'*Achat*. Il faut toujours choisir l'identifiant **minimum**.

L'identifiant de l'entité *Achat* est spécifié via la contrainte additionnelle :

id(Achat) : Client, Produit, Fournisseur, DateHeure

N.B. Il ne faut pas souligner l'attribut *DateHeure* sur le schéma ; il n'est en effet pas à lui seul identifiant de l'entité *Achat*.

2.10.3 Identifiant d'association implicite

① Une association **1 à N** est identifiée *implicitement* **par l'identifiant de l'entité** dont chaque occurrence ne participe qu'à une seule occurrence de l'association

Exemple :



Toute occurrence de l'association *Propriete* est un lien entre exactement **une occurrence de *Personne*** et **une occurrence de *Vehicule***. Comme un véhicule ne peut avoir qu'un propriétaire, il n'aura qu'un seul lien de type *propriete*. Toute occurrence de *Propriete* est par conséquent identifiée par le véhicule relié. Autrement dit, toute occurrence de *Propriete* est identifiée par l'attribut *NumeroChassis* du véhicule relié.

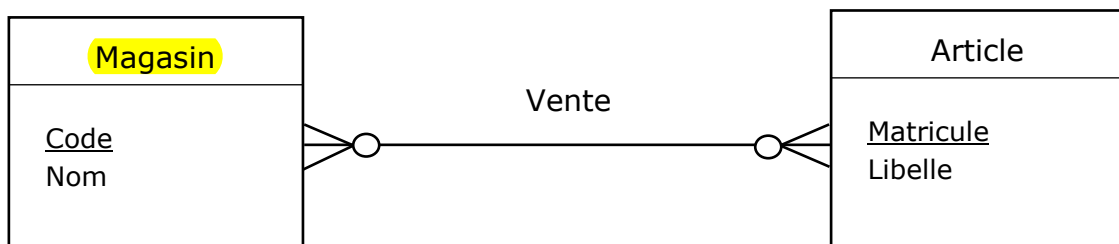
Le schéma ci-dessus contient **implicitement** la contrainte suivante :

Id(Propriete) : Vehicule

N.B. Puisque cette contrainte est implicite au schéma, il n'est donc pas nécessaire de l'ajouter explicitement dans les contraintes additionnelles.

② Une association **N à N** est identifiée *implicitement* **par la combinaison des identifiants des entités reliées.**

Exemple :



Toute occurrence de l'association *Vente* est un lien entre exactement **une occurrence de Magasin** et **une occurrence d'Article**.

Chaque occurrence de *Vente* est donc identifiée par la combinaison de l'identifiant du magasin relié d'une part et de l'identifiant de l'article relié d'autre part.

Soient l'occurrence de *Magasin* "*Le cellier du vigneron*" dont l'identifiant est "*CellVig*" et l'occurrence d'*Article* "*Château Lemaître 2019*" (bouteille de vin rouge de 75cl) dont l'identifiant est "*ChLem2019*". L'occurrence de l'association *Vente* qui relie le magasin "*Le cellier du vigneron*" et l'article "*Château Lemaître 2019*" est identifiée par la combinaison des identifiants "*CellVig*" et "*ChLem2019*". En effet, il n'y aura pas dans la base de données deux occurrences différentes de l'association *Vente* qui relieront l'occurrence de *Magasin* identifiée par "*CellVig*" et l'occurrence d'*Article* identifiée par "*ChLem2019*".

Le schéma ci-dessus contient **implicitement** la contrainte suivante :

Id(Vente) : Magasin, Article

N.B. Puisque cette contrainte est implicite au schéma, il n'est donc pas nécessaire de l'ajouter explicitement dans les contraintes additionnelles.

2.10.4 Autres contraintes additionnelles

Bien d'autres contraintes additionnelles peuvent être associées à un schéma entités associations, notamment des contraintes liées au métier.

Exemples :

On ne peut emprunter que 3 ouvrages simultanément

La date d'emprunt doit être inférieure ou égale à la date de retour.

Toutes les contraintes additionnelles liées au métier doivent bien évidemment être identifiées lors de l'étape d'analyse conceptuelle.

Cependant, **seules les contraintes additionnelles liées aux identifiants** seront exigées **lors de l'évaluation** du cours de Conception de bases de données du bloc 2.

2.11 Dépendances fonctionnelles

Comme expliqué précédemment, une source fréquente d'erreurs est la redondance des données dans les B.D. (cf section 2.7.).

Une façon de supprimer la redondance est de supprimer les dépendances fonctionnelles **anormales** entre attributs de la même entité.

Il y a une **dépendance fonctionnelle** entre un groupe d'attributs dits **déterminés** et un attribut **déterminant**, si les valeurs des attributs déterminés dépendent, sont fonction de la valeur du déterminant.

Exemple :

Etudiant
<u>Matricule</u>
Nom
Adresse

Dans l'entité *Etudiant*, les valeurs des attributs *Nom* et *Adresse* dépendent, sont fonction, de la valeur de l'attribut *Matricule*. Ou encore, deux occurrences d'*Etudiant* avec la même valeur de *Matricule* ont les mêmes valeurs pour *Nom* et *Adresse*.

On note une dépendance fonctionnelle comme suit :

Matricule → Nom, Adresse

Dans cette dépendance fonctionnelle, le déterminant est *Matricule* et les déterminés sont *Nom* et *Adresse*.

Les dépendances fonctionnelles **entre l'identifiant** de l'entité **et d'autres attributs** de la même entité sont des dépendances fonctionnelles **normales** qui n'engendrent **aucune redondance**.

Les dépendances fonctionnelles entre un **déterminant qui n'est pas l'identifiant** de l'entité **et d'autres attributs** de la même entité sont des dépendances fonctionnelles **anormales** qui engendrent de la **redondance**. Il faut donc éliminer ces dernières.

Autre exemple de dépendance fonctionnelle **normale** n'engendrant **pas de redondance** :

Personne
<u>Matricule</u> ... CodePostal Localite

Matricule → CodePostal, Localite

Pour rappel, ni le code postal ni le libellé de localité n'identifie à lui seul une localité. Cela n'engendre aucune dépendance fonctionnelle anormale.

~~CodePostal → Localite~~
~~Localite → CodePostal~~

Exemples de dépendances fonctionnelles **anormales engendrant de la redondance** :

Exemple 1 :

Etudiant
<u>Matricule</u> NomPrenom NomSection Coordinateur

NomSection → Coordinateur

Redondance : "Le coordinateur correspondant à une section donnée" est enregistré autant de fois qu'il y a d'étudiants dans cette section.

Exemple 2 :

Colis
<u>Numero</u> DateExpedition CodeBarreArticle PoidsArticle

CodeBarreArticle → PoidsArticle

N.B. On ne considère que les colis contenant un seul article.

Redondance : "Le poids de l'article correspondant à un code-barre donné" est enregistré autant de fois qu'il y a de colis contenant cet article.

Exemple 3 :

CoteExamen
<u>Numero</u>
MatriculeEtudiant
NomEtudiant
DateExamen
CodeUE
NbECTS
Resultat

MatriculeEtudiant → NomEtudiant

CodeUE → NbECTS

Redondances :

"Le nom d'étudiant correspondant à un matricule donné" est enregistré autant de fois que cet étudiant aura passé d'examens.

"Le nombre de crédits ECTS correspondant à une UE donnée" est enregistré autant de fois qu'il y a d'examens portant sur cette UE.

Exemple 4 :

Employe
<u>Matricule</u>
...
NumeroService
NomService
AdresseService

NumeroService → NomService, AdresseService

Redondance : "Le nom et l'adresse correspondant à un numéro de service donné" sont enregistrés autant de fois qu'il y a d'employés dans ce service.

Il existe des dépendances fonctionnelles dont le déterminant est composé de plusieurs attributs.

Exemple 5 :

Personne
<u>Matricule</u>
...
CodePostal
Localite
NbHabitants

CodePostal, Localite → NbHabitants

Cas souvent compliqué

F. Dubisy

Redondance : "Le nombre d'habitants correspondant à une localité donnée" est enregistré autant de fois qu'il y a de personnes reprises dans la B.D. habitant dans cette localité.

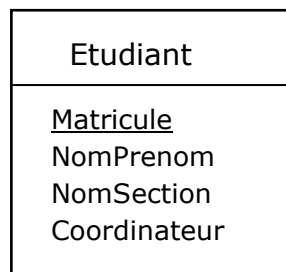
Pour supprimer la redondance interne engendrée par des dépendances fonctionnelles anormales, il faut **normaliser** l'entité.

Pour ce faire, on procède comme suit :

- On retire de l'entité tous les attributs intervenant dans la dépendance fonctionnelle anormale (déterminants et déterminés) et on en fait une nouvelle entité avec ces attributs dont l'identifiant sera le déterminant.
- On relie cette nouvelle entité à l'ancienne entité via une nouvelle association.

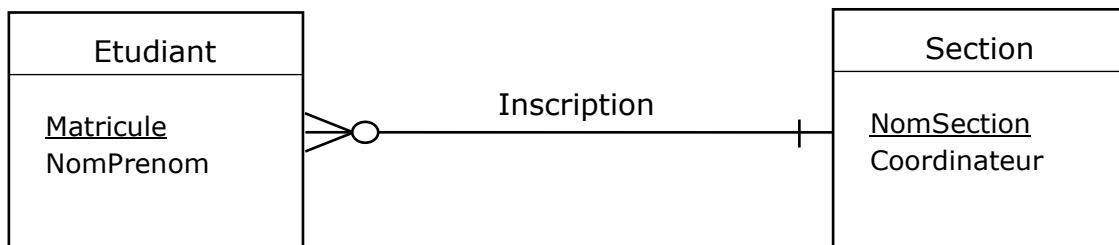
Exemple 1 :

Avec redondance :



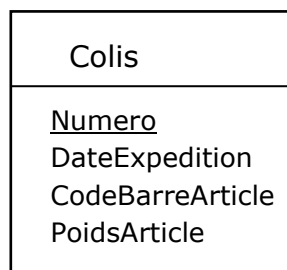
NomSection → Coordinateur

Sans redondance :



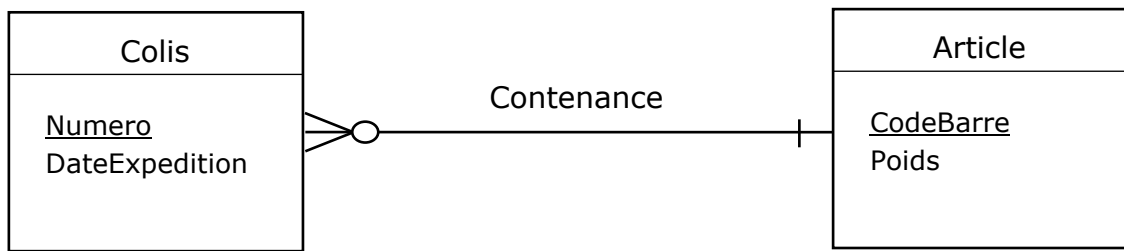
Exemple 2 :

Avec redondance :



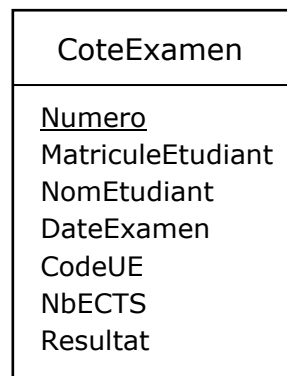
CodeBarreArticle → PoidsArticle

Sans redondance :



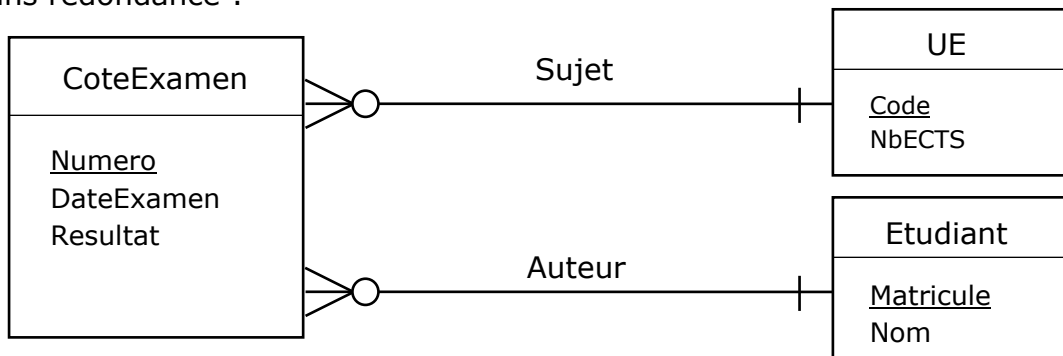
Exemple 3 :

Avec redondance :



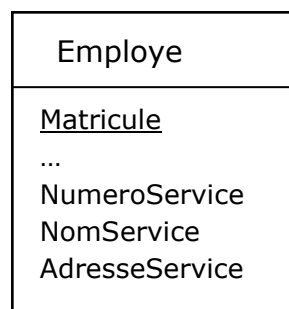
MatriculeEtudiant → NomEtudiant
CodeUE → NbECTS

Sans redondance :



Exemple 4 :

Avec redondance



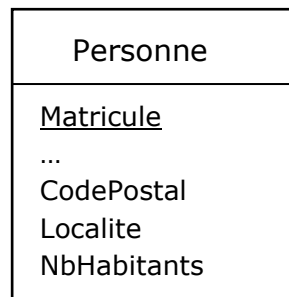
NumeroService → NomService, AdresseService

Sans redondance :



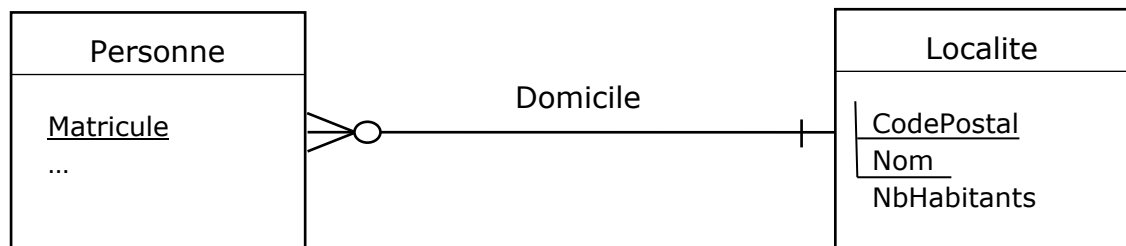
Exemple 5 :

Avec redondance



CodePostal, Localite → NbHabitants

Sans redondance :



2.12 Transformation de schéma

Il existe des transformations d'un schéma A en un schéma A' qui préservent la sémantique :

schéma A \Leftrightarrow schéma A'

Les deux schémas sont sémantiquement équivalents.

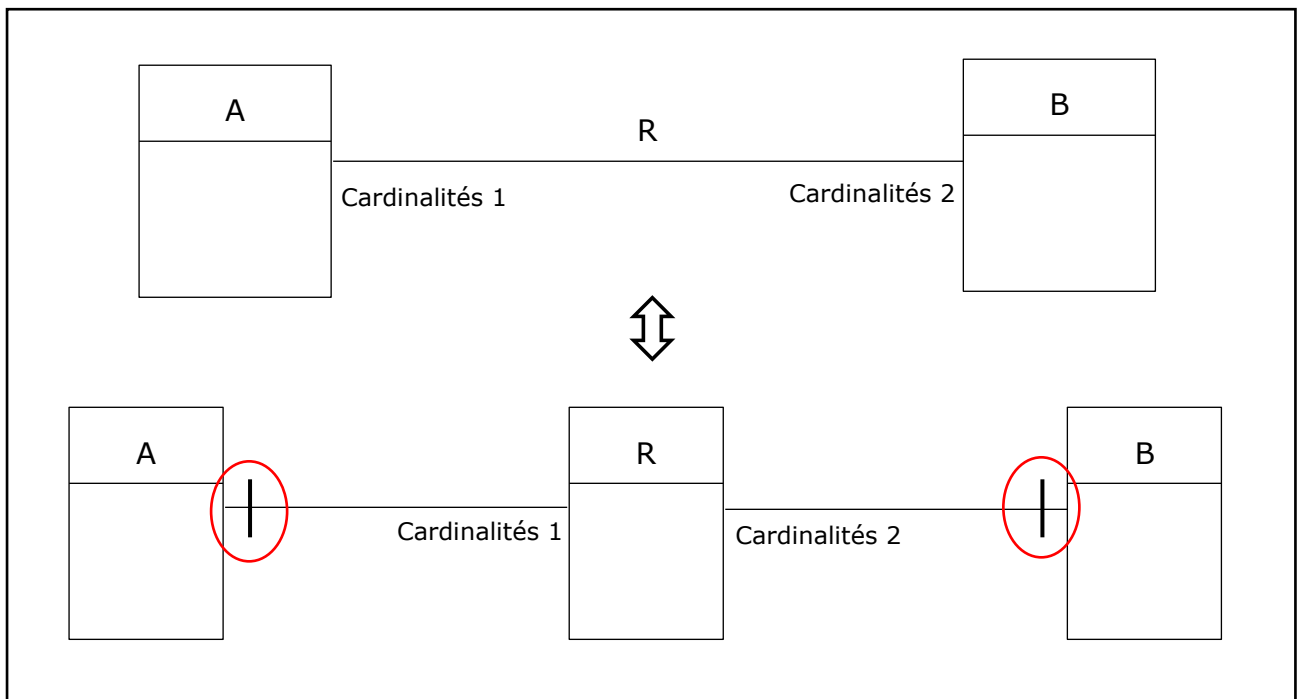
2.12.1 Transformation d'une association en une entité

On peut transformer toute association en une entité (+ deux nouvelles associations).

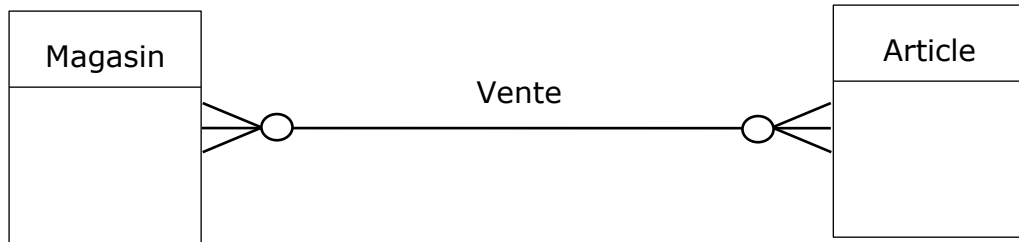
Les deux représentations sont équivalentes.

De plus, cette transformation est **réversible**.

Forme générale

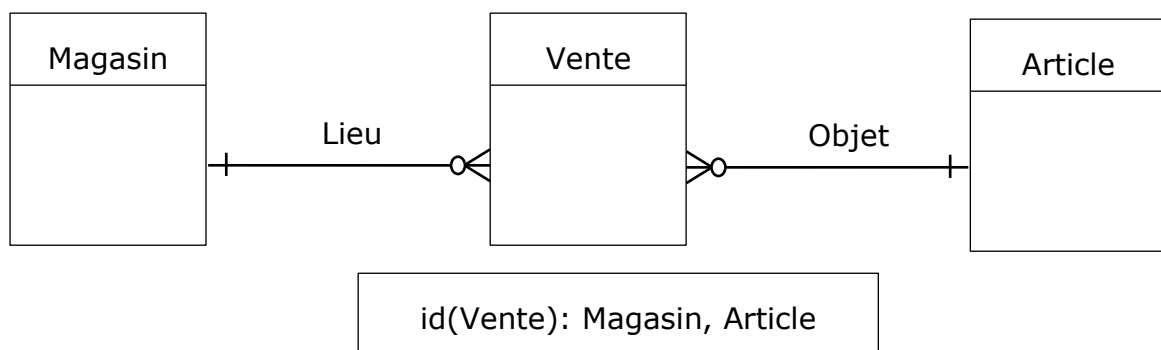


Exemple 1 : Relation binaire



Cette relation N à N contient un identifiant implicite : toute vente est identifiée par la combinaison des identifiants du magasin et de l'article associés.

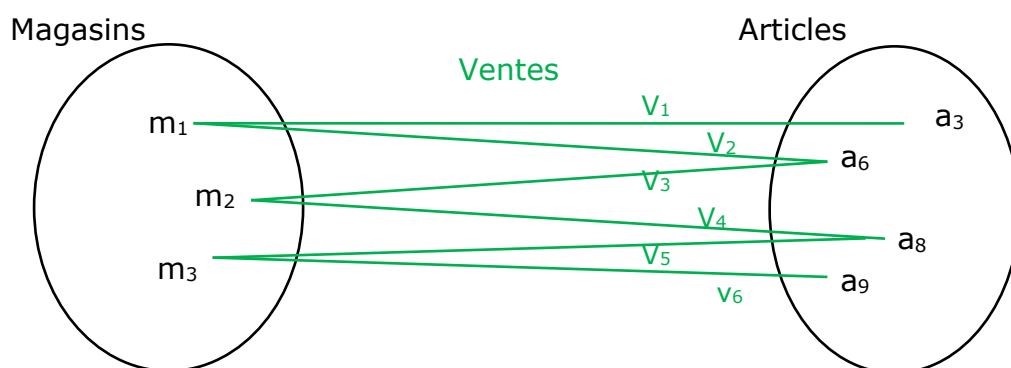
Ce schéma est équivalent à :



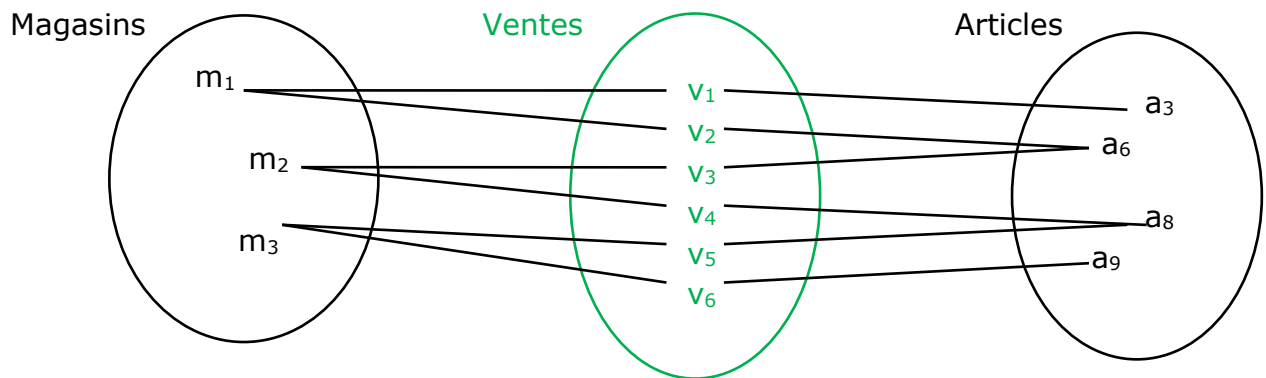
L'identifiant de la nouvelle entité *Vente* est donc la combinaison des identifiants des entités reliées, à savoir, *Magasin* et *Article*.

Au niveau occurrences :

① Via une association *Vente*



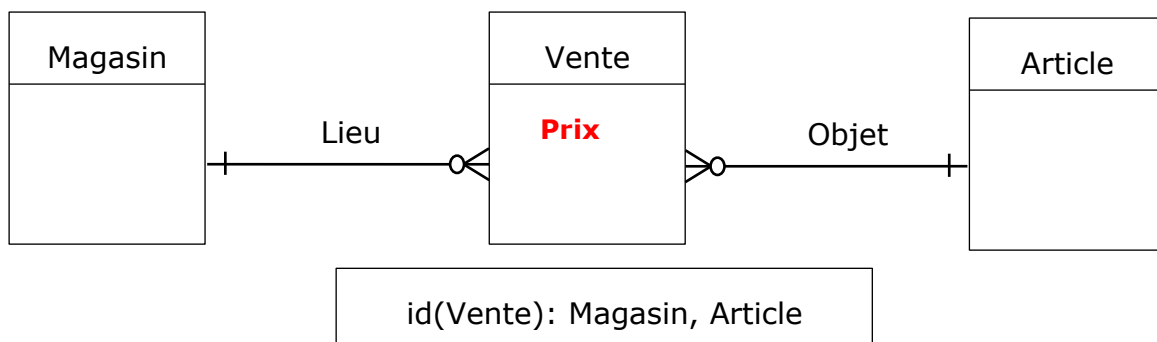
② Via une entité *Vente*



Dans quel cas est-il intéressant d'appliquer ce type de transformation ?

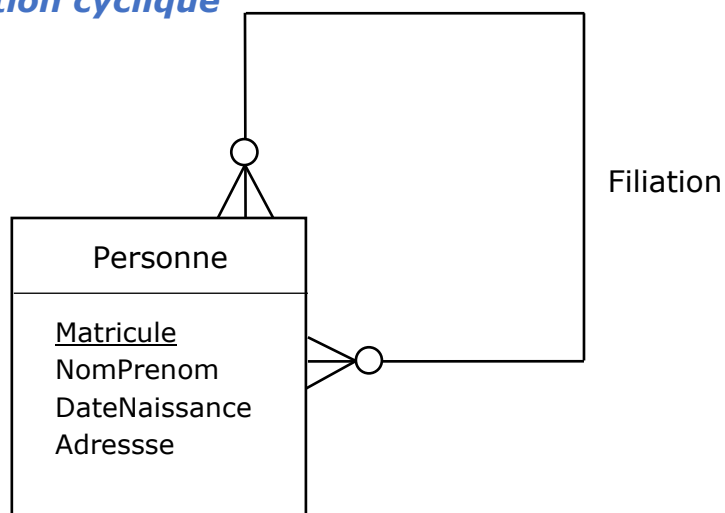
Par exemple s'il faut ajouter des caractéristiques, c'est-à-dire des attributs, à une association.

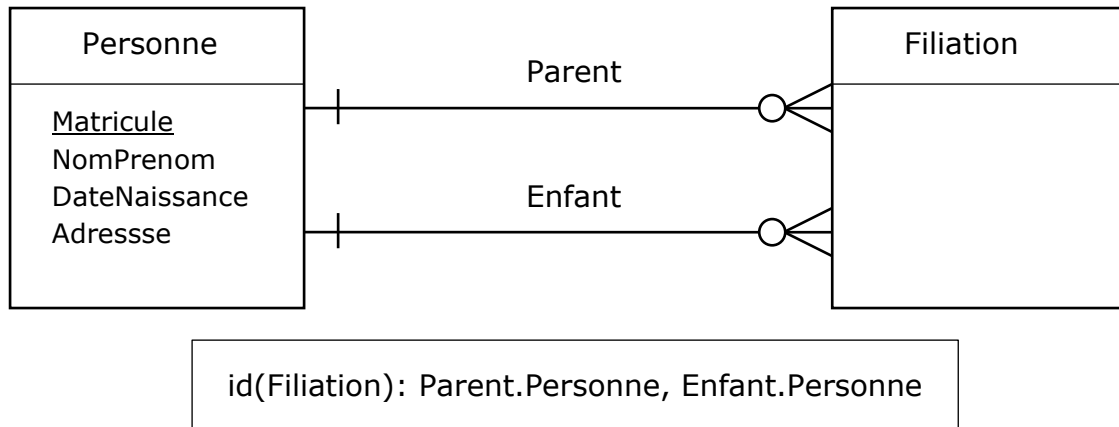
Exemple, il faut ajouter le prix à l'association *Vente*.



A noter que l'identifiant ne change pas ; l'attribut *Prix* ne fait pas partie de l'identifiant minimum.

Exemple 2 : Association cyclique

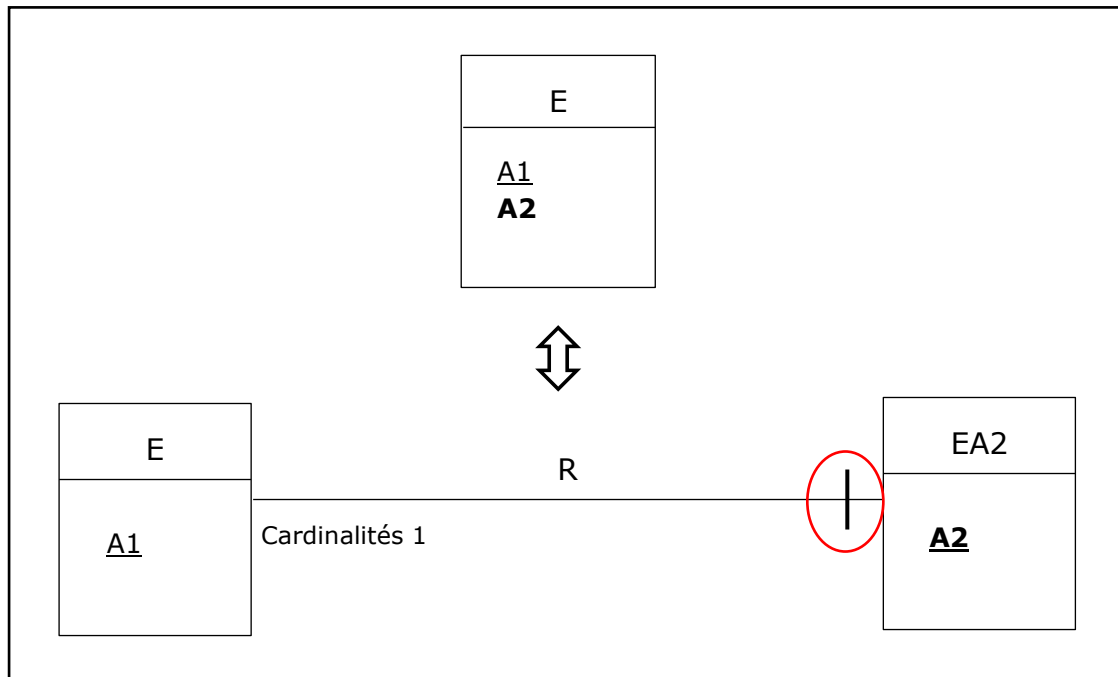




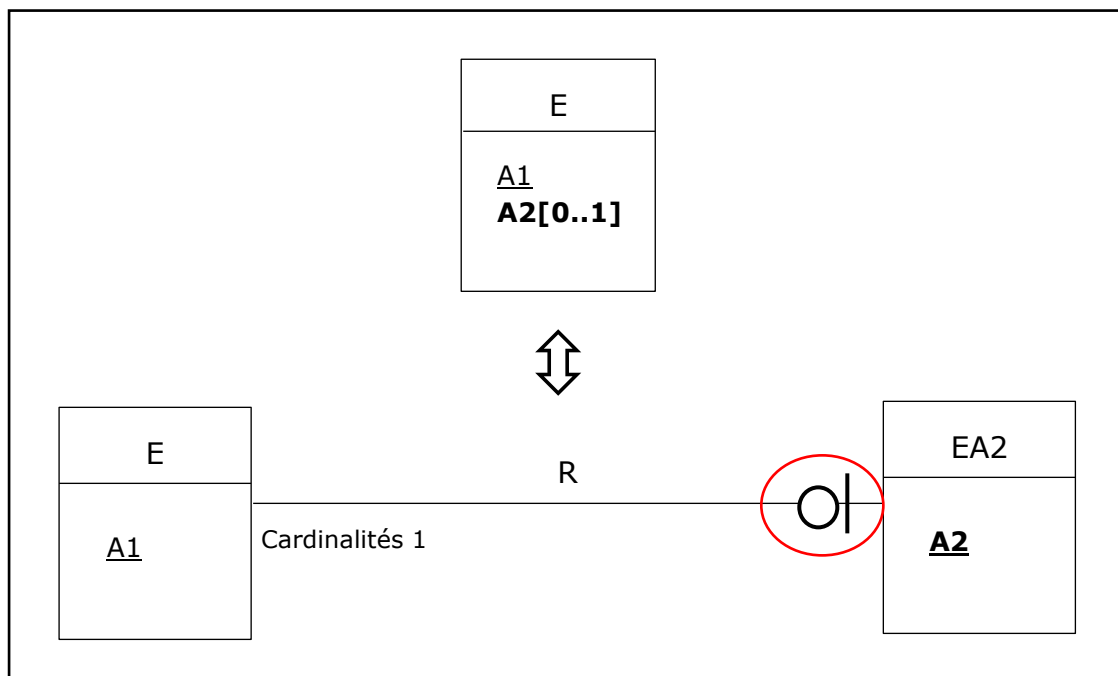
A noter que cette version a l'avantage de permettre de qualifier toute filiation ; on pourrait en effet ajouter des attributs à l'entité *Filiation* précisant par exemple le type de filiation (paternité, maternité, adoption...) ou si les deux personnes s'entendent ou non (via un booléen).

2.12.2 Transformation d'attribut en entité

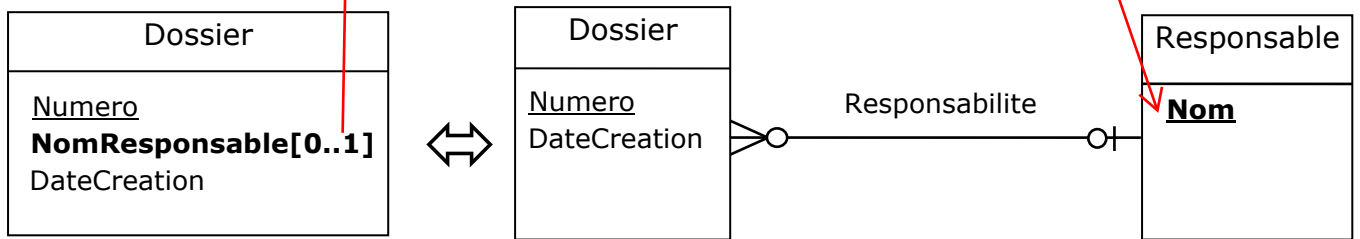
Attribut obligatoire



Attribut facultatif



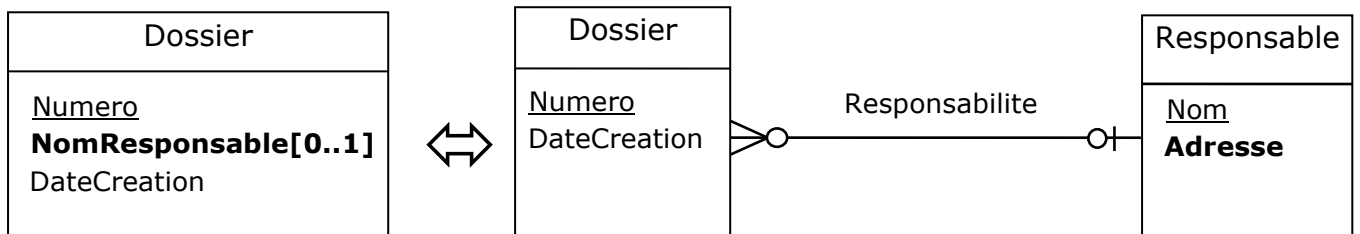
Exemple :



Utilité

① Quand on se rend compte qu'un **attribut représente un concept à part entière**, on en fait une entité. C'est par exemple le cas quand on doit lui ajouter une caractéristique supplémentaire.

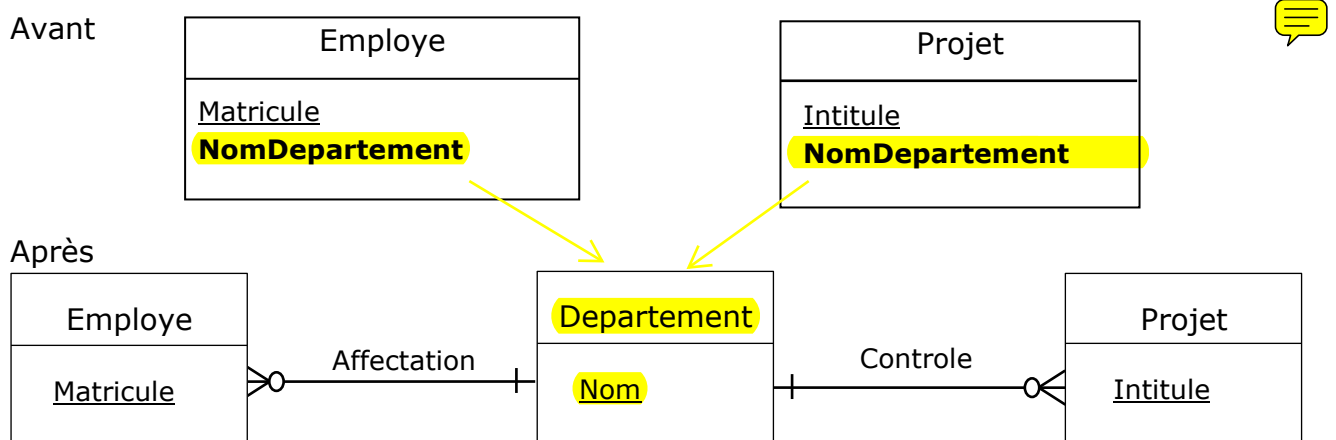
Exemple : On veut ajouter la notion d'adresse au responsable de chaque dossier.



② Quand **le même attribut est cité à plusieurs endroits**, il peut s'agir d'un concept à part entière à isoler, c'est-à-dire en faire une nouvelle entité.

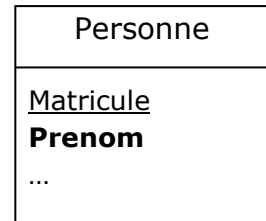
Exemple : L'attribut *NomDepartement* apparaît dans

- l'entité *Employe*, car un employé est affecté à un département ;
- l'entité *Projet*, car un projet est contrôlé par un département.

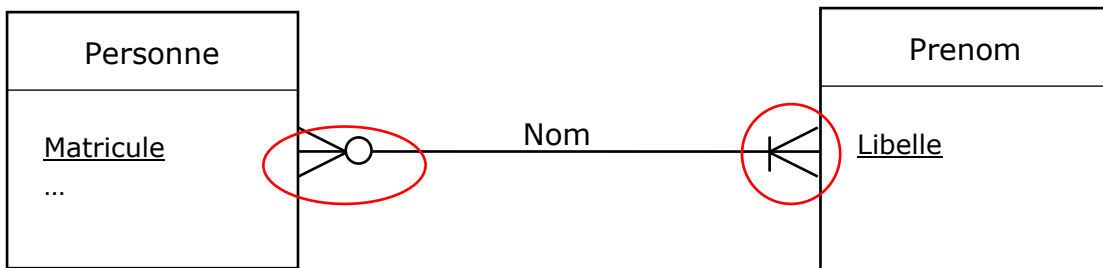


③ Quand un attribut peut prendre plusieurs valeurs pour la même occurrence d'entité.

Exemple : Une personne ne peut avoir qu'un seul prénom.



Si une personne peut avoir plusieurs prénoms, il faut transformer le schéma :



Partie 3

Analyse logique

3 Bases de données relationnelles

Beaucoup de bases de données actuelles sont de type relationnel.

Le modèle relationnel a été introduit par Codd dès 1970. Ce modèle est basé sur une structure unique et uniforme : la **relation**.



Le langage d'exploitation des bases de données relationnelles le plus connu est **SQL**, pour **S**tructured **Q**uery **L**anguage. SQL a été développé par IBM.

Ce n'est pas un langage de programmation à proprement parlé (SQL est non procédural), mais un langage de requêtes basé sur la théorie des ensembles. Pour interroger la base de données, on n'indique plus au S.G.B.D. **comment** retrouver l'information, on se contente de spécifier ce qu'on veut obtenir (le **quoi**). On parle encore de langage **déclaratif**.



Un standard SQL a été défini conjointement par ANSI (American National Standards Institute) et ISO (International Standards Organization). Ce standard est implémenté en différents "dialectes" par les S.G.B.D. relationnels (*Oracle*, *MySQL*, *SQL Server*, *SQLite...*) ; ces "dialectes" présentent donc quelques différences syntaxiques.



3.1 Structure de base : les tables

Les bases de données relationnelles tiennent leur nom de la structure de base qu'est la **relation**. Une relation est une structure qui rassemble **des données reliées** entre elles.

La relation est une **table de valeurs**.

3.1.1 Table

une entité va devenir une table

Une table représente **un concept** qui peut être :

	un objet ,	une personne ,	un fait ,	une situation ...
	⇓	⇓	⇓	⇓
Exemples :	<i>voiture</i>	<i>propriétaire</i>	<i>emprunt</i>	<i>accident</i>
	<i>ouvrage</i>	<i>client</i>	<i>construction</i>	
	<i>article</i>	<i>emprunteur</i>		
	<i>magasin</i>	<i>étudiant</i>		

Une table contient des lignes et des colonnes.

3.1.2 Ligne = 1 occurrence de l'entité

Une ligne correspond à **une occurrence** du concept.

Ainsi, une ligne de la table *client* reprend toutes les informations d'un client particulier, par exemple, le *client Dupond* :

Dupond	Georges	32, rue de Fer, 5000, Namur	081/21.22.23	24/12/1963	marié
--------	---------	-----------------------------	--------------	------------	-------

Les données d'une même ligne sont en **relation** les unes avec les autres ; elles forment un tout. D'où, le nom de relation pour une table.



💡 Il ne faut toutefois **pas confondre relation (ou table) avec la notion d'association** du schéma conceptuel. Nous verrons plus loin comment représenter en relationnel les associations.

3.1.3 Colonne = 1 attribut de l'entité

Une colonne correspond à **un attribut** du concept.

Les attributs permettent d'exprimer le rôle joué par chacune des valeurs d'une même ligne.

Pour chaque attribut/colonne, on définit un **nom**, un **type**, une **longueur**, et éventuellement un **domaine** des valeurs permises.



💡 Il est **possible d'ajouter, modifier ou supprimer des colonnes après création de la table.**

Une colonne peut être obligatoire ou facultative. Si elle est facultative, elle sera notée [0..1].

Une colonne peut être facultative

NB : on peut rendre une valeur **obligatoire** -> **facultative** mais on ne peut **PAS** rendre une valeur **facultative** -> **obligatoire** (à cause du manque d'informations)

valeur particulière = NULL (= valeur inconnue (=/= 0))

3.1.4 Valeur

La valeur est donnée par l'intersection d'une ligne et d'une colonne. Il s'agit de la valeur d'un attribut pour une occurrence particulière de la table.

Plaque	Marque	Modele	DateFabrication	Kilometrage
1-MBX-935	Renault	Espace	12/08/2010	55.897

Il existe une valeur particulière, à savoir, la valeur **null**, à ne pas confondre avec la valeur 0 pour les colonnes de type numérique ou la chaîne de caractères à blanc pour les colonnes de type alphanumérique. Une ligne particulière peut prendre la valeur null pour un attribut particulier. La valeur null peut avoir trois significations :

Les 3 significations de NULL (jamais 0) :

- ① La valeur de l'attribut est **inconnue** pour certaines occurrences.
Exemple : le numéro de compte d'un client particulier est inconnu (différent de la valeur 0 si l'attribut numéro de compte est défini de type numérique)
- ② L'attribut ne **s'applique pas à un groupe d'occurrences**.
Exemple : nom d'épouse pour les hommes
- ③ Certaines occurrences ne **possèdent pas de valeur pour l'attribut**.
Exemple : numéro de téléphone pour les personnes sans téléphone

3.2 Notion d'ordre

L'ordre de lignes se met à l'affichage

Les lignes d'une table ne sont pas ordonnées

=> **pas de notion d'ordre au niveau logique**

3.2.1 Ordre des lignes

Une table est un ensemble de lignes. La notion d'ensemble est à prendre au sens mathématique. Les éléments d'un ensemble ne sont pas ordonnés. Les lignes d'une table ne sont donc pas ordonnées. Même si au niveau physique, les lignes d'une table sont stockées dans un fichier, et par conséquent dans un certain ordre (physique), il n'y a **pas de notion d'ordre au niveau logique**.

Il n'est donc pas possible d'accéder à la première ligne, à la 5ème ligne ou à la dernière ligne d'une table.

De même, l'ordre des lignes sélectionnées par une requête est non significatif : demander les voitures de marque *Toyota* fabriquées *entre 2010 et 2020* produira une liste. La même requête exécutée ultérieurement pourrait produire la liste des mêmes voitures mais retournées dans un ordre différent. Ce sera le cas par exemple, si la structure de la table a été modifiée (ajout de nouvelles contraintes...).

Il est cependant possible de spécifier dans une requête l'ordre dans lequel on veut voir s'afficher les lignes qui satisfont la requête (cf la clause **order by** en SQL).

Notons que le nombre de lignes est souvent beaucoup plus élevé que le nombre de colonnes.

Les lignes sont dynamiques, dans le sens où de nouvelles lignes peuvent être insérées et des lignes existantes modifiées ou supprimées.

Les colonnes, quant à elles, sont plus statiques : le nombre de colonnes est relativement fixe. Bien que cela soit possible en relationnel, on ne rajoute de nouvelles colonnes ou on ne modifie la définition de colonnes existantes que rarement.

3.2.2 Ordre des colonnes

Il en va de même **pour les colonnes comme pour les lignes : aucun ordre n'existe entre les colonnes.**



L'ordre des colonnes en résultat à une requête ne dépend pas de l'ordre de création des colonnes dans la table, mais de l'ordre des colonnes tel qu'il est spécifié dans la requête elle-même.

Par exemple, les requêtes

`select Nom, Prenom, DateNaissance, Statut, Telephone from Personne (1)`

et `select Prenom, Nom, Telephone, Statut, DateNaissance from Personne (2)`

ne retourneront pas le même résultat.

Requête (1) :

Nom	Prenom	DateNaissance	Statut	Telephone
Dupond	Albert	12/12/1988	marié	081/21.33.12
Patrice	Paul	01/08/2000	célibataire	02/732.25.26
Perssin	Claude	20/02/1976	marié	071/81.12.40
...

Requête (2) :

Prenom	Nom	Telephone	Statut	DateNaissance
Albert	Dupond	081/21.33.12	marié	12/12/1988
Paul	Patrice	02/732.25.26	célibataire	01/08/2000
Claude	Perssin	071/81.12.40	marié	20/02/1976
...

En théorie, en SQL, un identifiant n'est PAS obligatoire mais dans le cours il sera TOUJOURS demandé

3.3 Identifiant ou clé primaire

La **clé primaire** d'une table est utilisée pour **identifier** de façon univoque chaque ligne de la table. A chaque valeur de la clé primaire correspond une et une seule ligne, et à une ligne de la table correspond une et une seule valeur de clé primaire.

Une clé primaire (**primary key** en SQL) est composée d'**un ou plusieurs attributs**, c'est-à-dire d'une ou plusieurs colonne(s).

Le langage SQL n'impose pas que toutes les tables aient un identifiant. Les tables pour lesquelles aucune clé primaire n'est déclarée peuvent contenir alors plusieurs lignes tout à fait identiques.

Rappelons qu'une table n'étant pas ordonnée, il est impossible de demander l'accès à la première ligne, à la 5ème ligne, etc. **On utilisera donc la clé primaire pour sélectionner une ligne spécifique.** Si le numéro d'emprunteur est la clé primaire (identifiant) de la table *Emprunteur*, on pourra, par exemple, accéder à la ligne concernant l'emprunteur ayant pour identifiant le numéro *310*.

Un bon identifiant est un **identifiant invariant**. La combinaison des colonnes constituant la clé primaire aura donc une **valeur fixe** pendant toute la durée de vie d'une même ligne.

Par convention, le nom de la ou des colonnes constituant la clé primaire sera **souligné** dans la table.

Exemple :



<u>CodePostal</u>	<u>Libelle</u>	NombreHabitants
5020	Vedrin	6691
5020	Daussoulx	652
5020	Malonne	5256
6230	Buzet	4102
5530	Buzet	569
...

Une clé primaire aura TOUJOURS une valeur



☛ Puisque le rôle d'une clé primaire est d'identifier toute ligne de la table, **aucune clé primaire ne peut prendre la valeur *null***. En effet, si plusieurs lignes de la table avaient la valeur *null* pour clé primaire, aucune de ces lignes ne pourrait plus être identifiée de façon univoque par la clé primaire.

Si une même table possède **plusieurs identifiants**, un seul sera choisi pour jouer le rôle de clé primaire. En effet, en langage SQL, une seule combinaison de colonnes peut être déclarée **primary key**. D'autres identifiants peuvent cependant être spécifiés ; ils seront déclarés **clés candidates ou alternatives** (clause **unique** en SQL).

En SQL on peut avoir plusieurs identifiants mais 1 seule clé primaire parmi ceux-ci

3.4 Traduction d'un schéma conceptuel en relationnel

3.4.1 Représentation des entités et des attributs

Une **entité** est représentée en base de données relationnelle par une **table**.

Un **attribut** est représenté par une **colonne**. Une colonne peut être **facultative**. Les attributs facultatifs sont donc représentés par des colonnes facultatives.

Une table peut être présentée dans un schéma logique soit horizontalement, soit verticalement.

Exemple : Soit l'entité *Personne*

Personne
<u>NumeroRegistreNational</u>
Nom
NomEpouse [0..1]

L'entité *Personne* est traduite en une table *Personne* en schéma logique relationnel. Soit version verticale, soit horizontale

⇒ La table *Personne* présentée horizontalement :

Version
horizontale

Personne		
<u>NumeroRegistreNational</u>	Nom	NomEpouse[0..1]

⇒ La table *Personne* présentée verticalement :

version verticale
(qui sera la + utilisée
dans le cours à cause de
la place qu'elle prend)

Personne
<u>NumeroRegistreNational</u>
Nom
NomEpouse [0..1]

N.B. Pour des raisons pédagogiques, les tables seront présentées horizontalement dans ce syllabus. Par contre, en pratique, les schémas logiques qui présentent les tables verticalement sont plus lisibles, surtout si le nombre de table augmente. C'est pourquoi, dans les séances d'exercices (labos) les schémas logiques seront présentés verticalement.

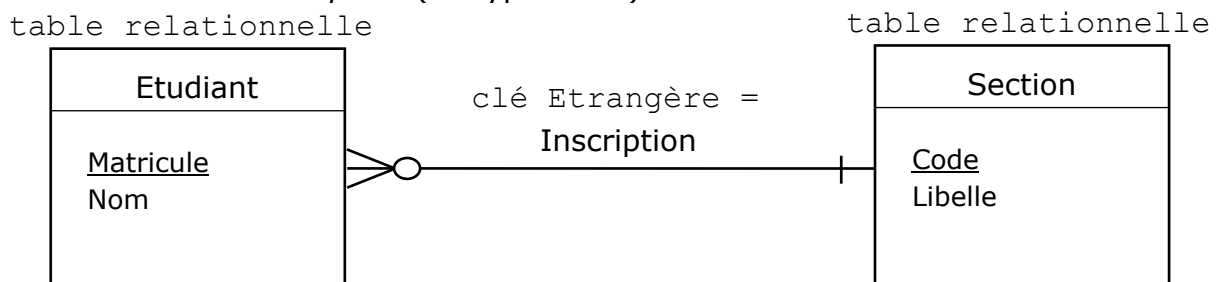
3.4.2 Représentation des associations

A. Représentation des associations 1 à N

= Règle 1

La notion de **clé étrangère** permet de représenter une relation de type **1 à N** entre deux tables.

Soit l'association *Inscription* (de type 1 à N) entre les entités *Etudiant* et *Section*.



Les entités *Etudiant* et *Section* sont toutes deux représentées par des tables en relationnel. L'association *Inscription* est représentée en relationnel par une **clé étrangère**.

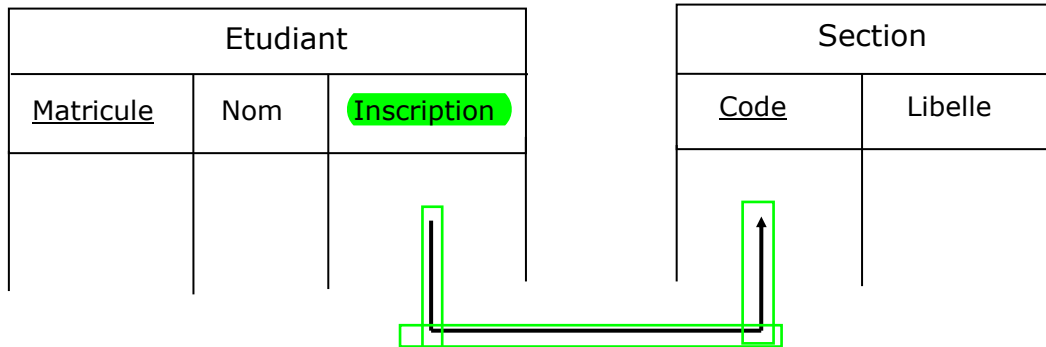
Une clé étrangère est une **colonne supplémentaire** que l'on ajoute à la table dont chaque occurrence ne participe qu'à une seule occurrence de l'association.

Cette nouvelle colonne fait référence à la table que l'on veut relier : elle jouera le rôle de **charnière** entre les deux tables. Chaque valeur de la colonne charnière doit référencer à coup sûr **une et une seule ligne** de la table référencée.

Par conséquent, **la colonne qui joue le rôle de clé étrangère doit référencer l'identifiant de la table que l'on veut relier.**

Dans l'exemple de la relation 1 à N *Inscription*, un étudiant ne peut être inscrit que dans une seule section. Tout étudiant doit donc avoir une référence vers l'identifiant de la section dans laquelle il est inscrit !

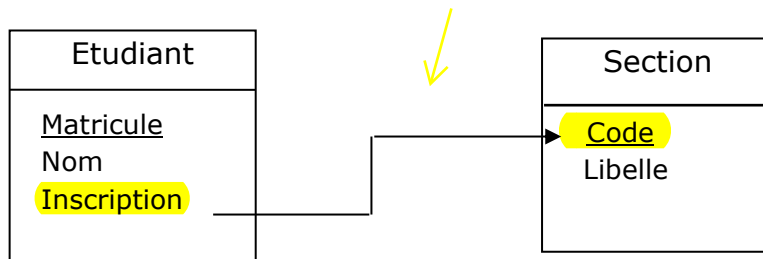
⇒ On **ajoute une colonne charnière (clé étrangère) dans la table *Etudiant***, qui reprendra pour chaque étudiant **l'identifiant** de sa section.



La nouvelle colonne intitulée *Inscription* doit être déclarée **de même type que l'identifiant** de la table *Section*, à savoir *Code*.

La clé étrangère peut être symbolisée sur le schéma logique (schéma des tables) par une **flèche** qui part de la colonne charnière vers l'identifiant de la table référencée.

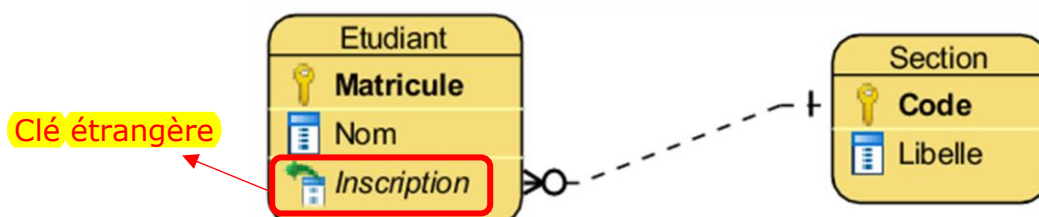
Ou encore, si l'on présente les **tables verticalement** :



Le nom à donner à la colonne clé étrangère est laissé à l'appréciation du concepteur de la base de données et ne doit obéir à aucune règle. La colonne (clé étrangère) intitulée dans l'exemple *Inscription* aurait pu tout aussi bien être intitulée *Section*.

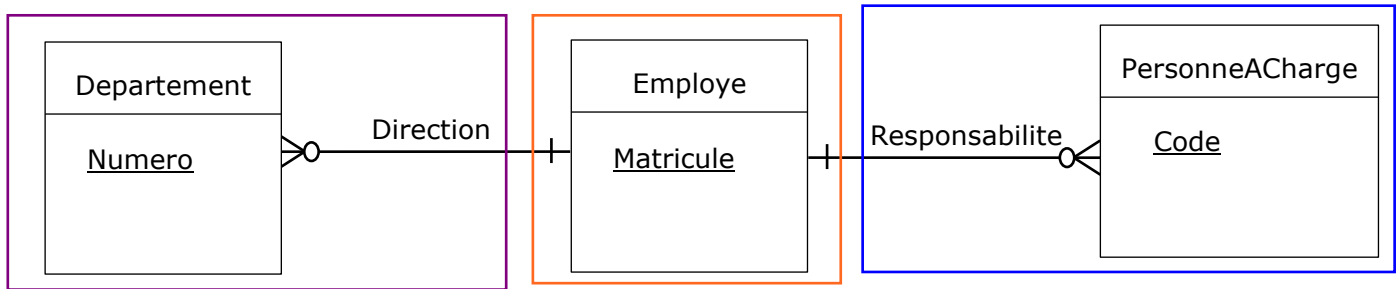
Notons que le principe des clés étrangères est un principe qui découle du bon sens. En effet, il est impossible de prévoir une colonne additionnelle dans la table *Section* en vue d'implémenter la relation 1 à N *Inscription*. A une section, il faudrait associer plusieurs occurrences d'étudiants. Plusieurs colonnes devraient par conséquent être ajoutées à la table *Section*, une colonne par étudiant inscrit dans la section ! Ceci est ingérable : combien de colonnes prévoir ?

Notation en Visual Paradigm :

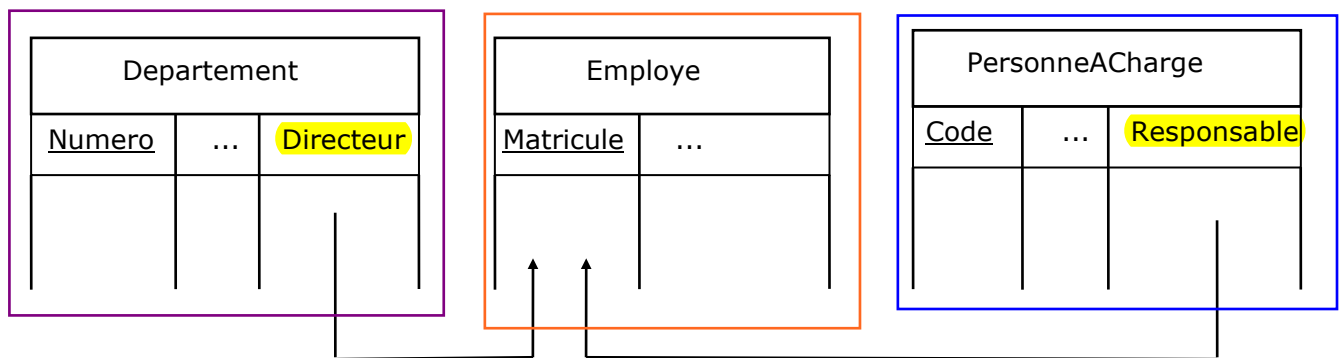


Exemple 1 :

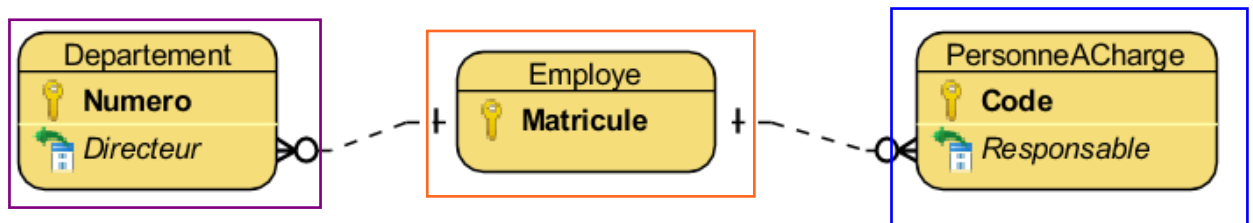
Soit le schéma entités-associations suivant :



La traduction logique en base de données relationnelle du schéma conceptuel ci-dessus est la suivante :

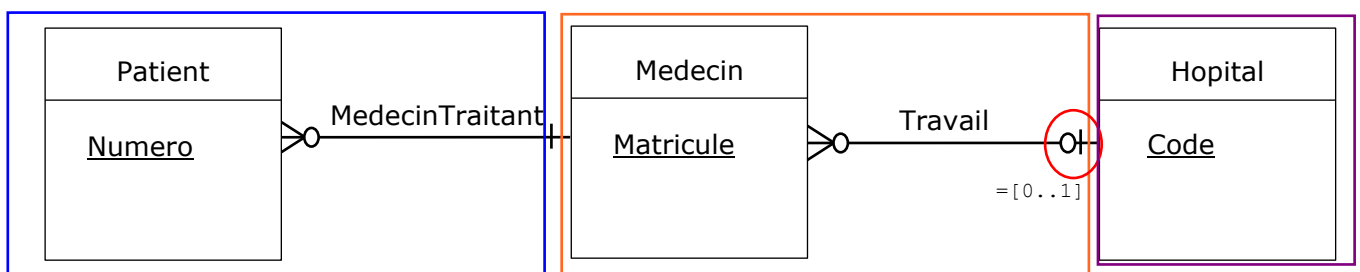


En Visual Paradigm :

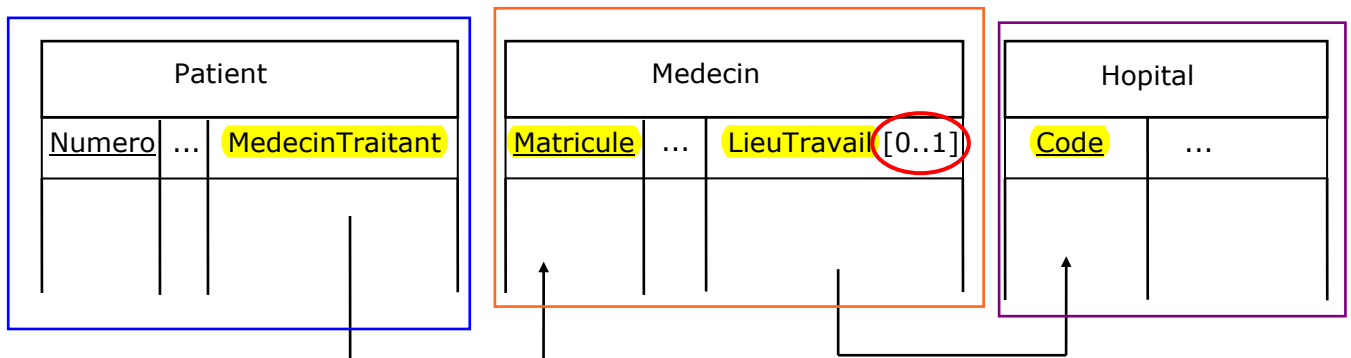


Exemple 2 :

Soit le schéma entités-associations suivant :



La traduction logique en base de données relationnelle du schéma conceptuel ci-dessus est la suivante :



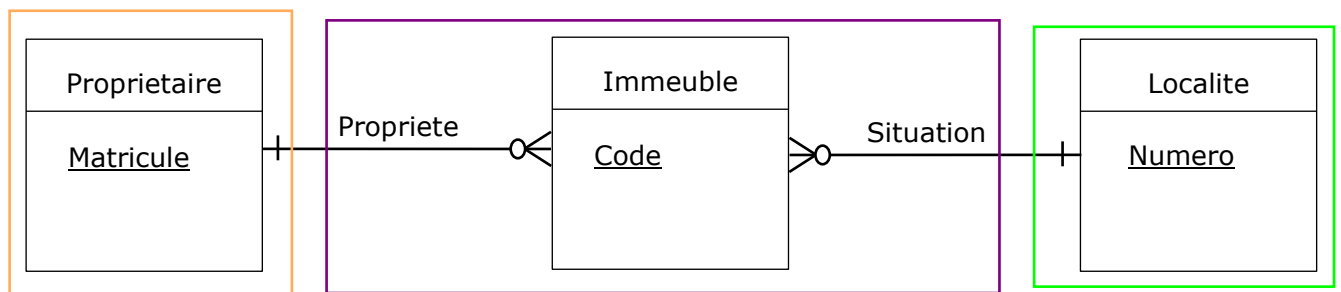
Une **clé étrangère** peut donc être **facultative**.

En Visual Paradigm :

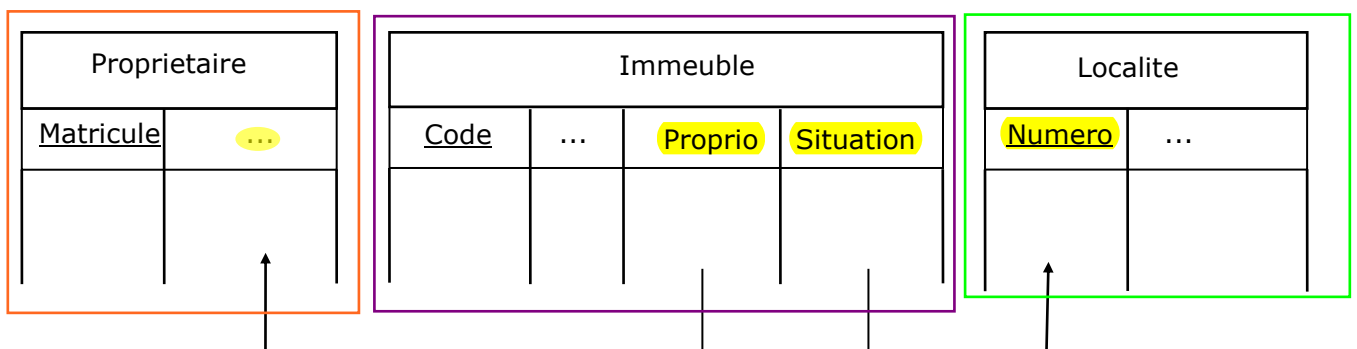


Exemple 3 :

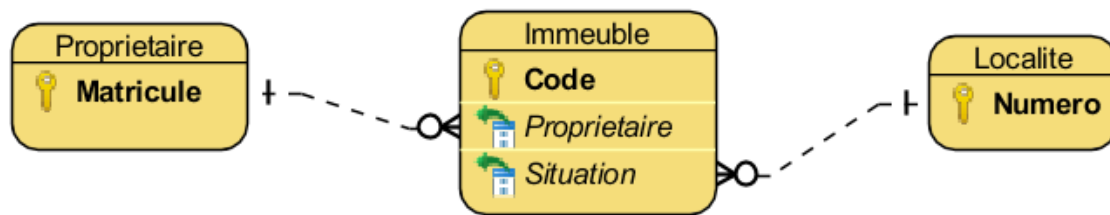
Soit le schéma entités-associations suivant :



La traduction logique en base de données relationnelle du schéma conceptuel ci-dessus est la suivante :



En Visual Paradigm :

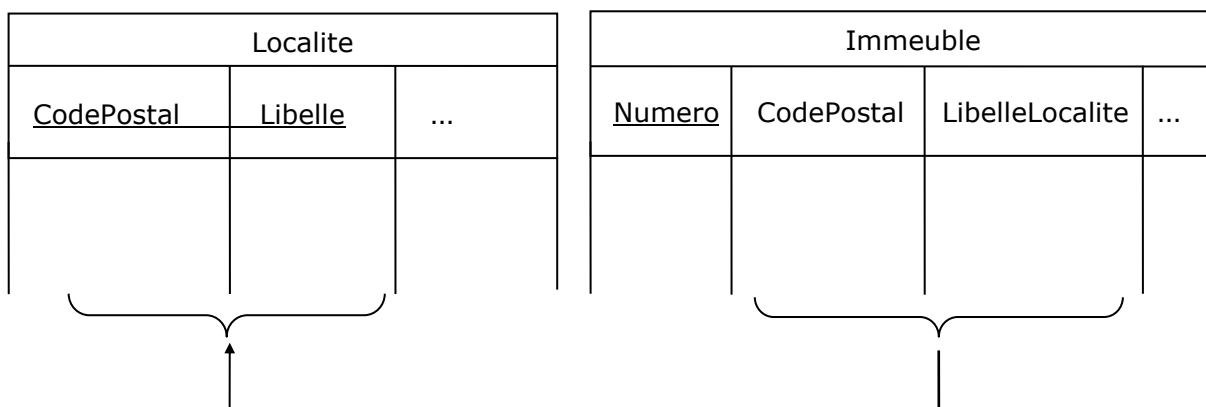


Si l'identifiant de la table reliée est composé de plus d'une colonne, la clé étrangère sera composée d'autant de colonnes.

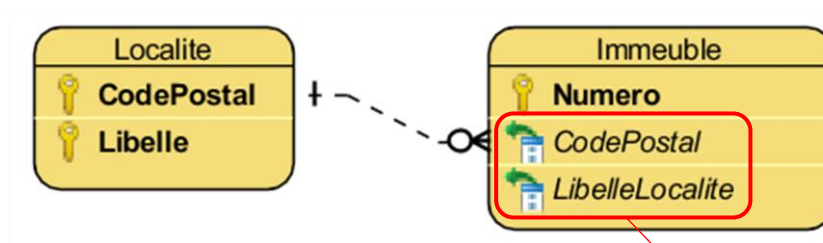
Exemple 4 : Soit le schéma entité-association suivant :



La traduction logique en base de données relationnelle du schéma conceptuel ci-dessus est la suivante :



En Visual Paradigm :



Une seule clé étrangère composée de deux colonnes

B. Représentation des associations N à N

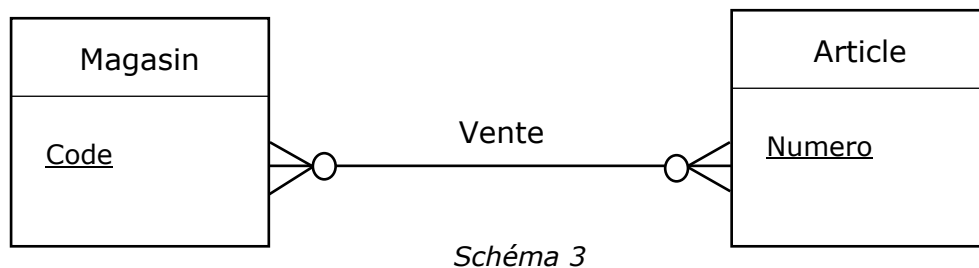
= Règle 2

Seules les associations 1 à N peuvent être traduites directement en relationnel : une relation 1 à N est représentée par une clé étrangère. Il n'en va pas de même avec les associations N à N. Il faut passer par une étape intermédiaire et effectuer une transformation du schéma entité-association (cf. section 2.12.1) : il faut **éclater toute association N à N en deux associations 1 à N**, qui à leur tour seront représentées en relationnel par des clés étrangères.

Une association **N à N** est donc représentée par une **table supplémentaire**. La traduction d'une association N à N entre deux entités aboutit donc à la création de trois tables, une table pour chaque entité et une table pour l'association N à N.

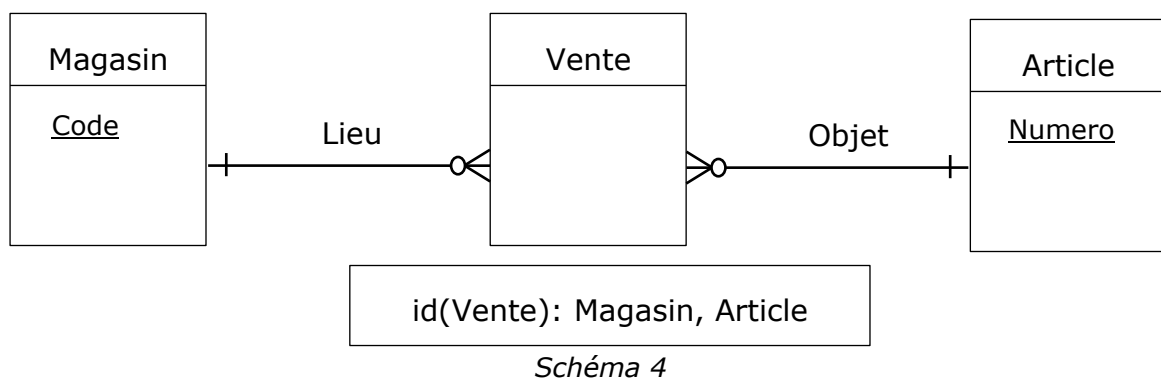
Exemple 1

Soit l'association *Vente* entre les entités *Magasin* et *Article*



Pour rappel, cette relation N à N contient un identifiant implicite : toute vente est identifiée par la combinaison des identifiants du magasin et de l'article associés.

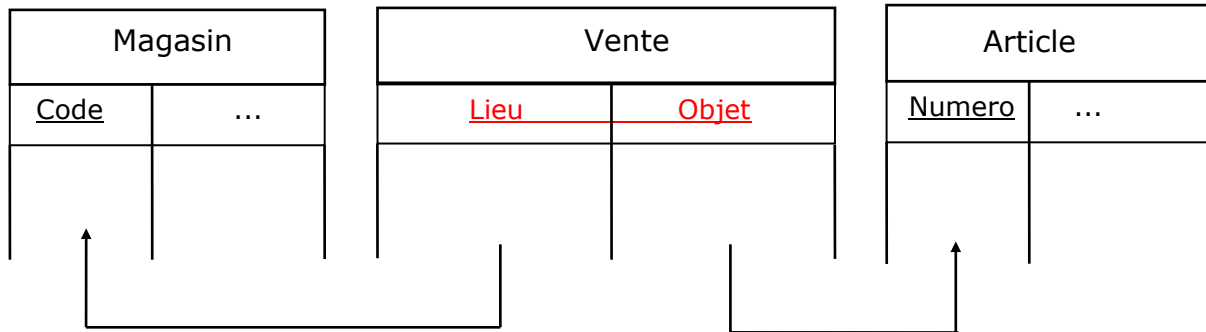
L'association *Vente* peut être considérée comme un concept et non plus comme un lien. Une transformation de ce schéma conceptuel peut être proposée : les schémas conceptuels (3) et (4) sont sémantiquement équivalents.



donc faire une id hybride => nouvelle table (ici Vente)

L'association N à N a été éclatée en deux associations 1 à N. Les trois entités *Magasin*, *Article* et *Vente* seront représentées respectivement par les trois tables *Magasin*, *Article* et *Vente*. Les deux associations *Lieu* et *Objet* seront représentées par deux clés étrangères au sein de la table *Vente*.

La table *Vente* ne sera en fait constituée que de deux colonnes, les deux clés étrangères.



L'identifiant de la table *Vente* est composé des deux colonnes clés étrangères.

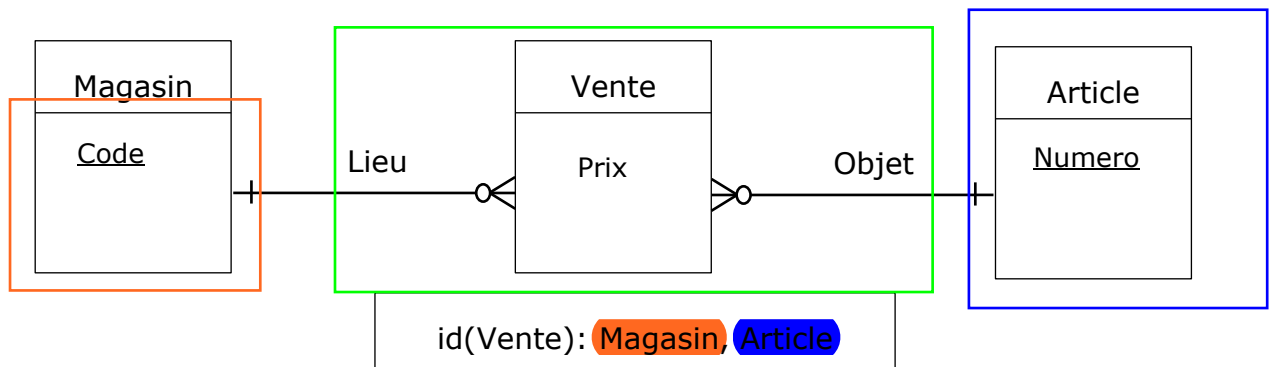
En Visual Paradigm :



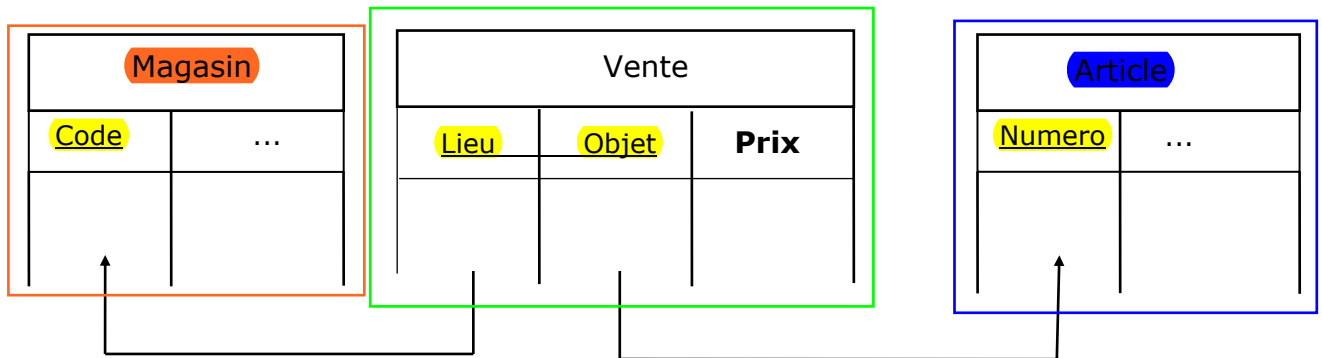
Symbolisme pour clés étrangères faisant partie de l'identifiant

L'entité *Vente* peut contenir des **attributs** propres ; ceux-ci constitueront autant de colonnes supplémentaires dans la table *Vente*.

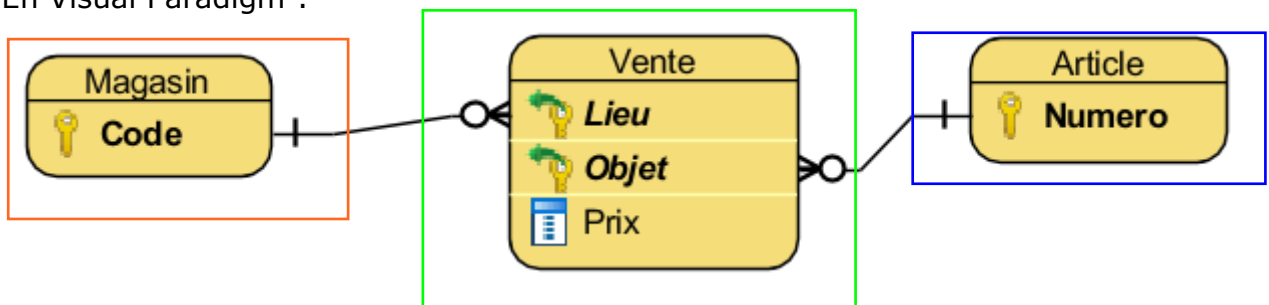
Exemple 2



La traduction logique en base de données relationnelle du schéma conceptuel ci-dessus est la suivante

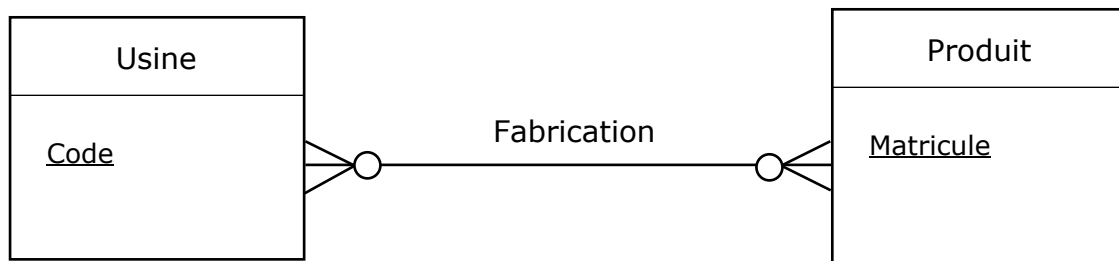


En Visual Paradigm :

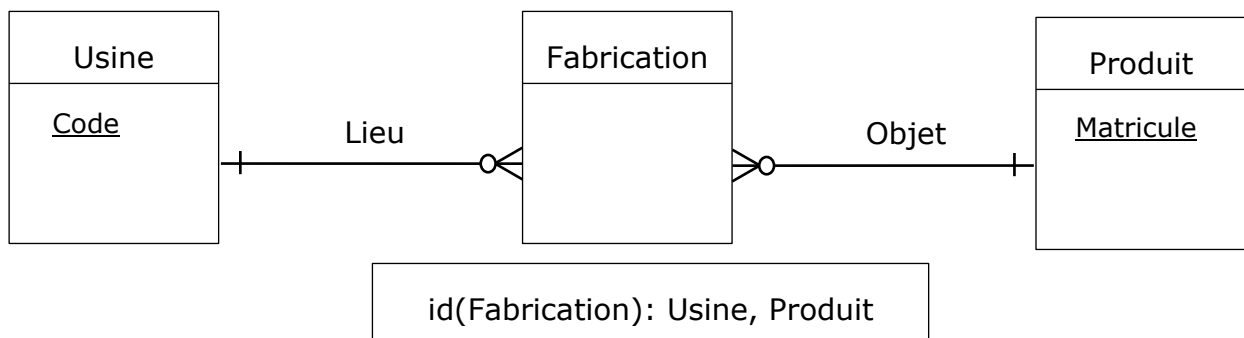


Exemple 3

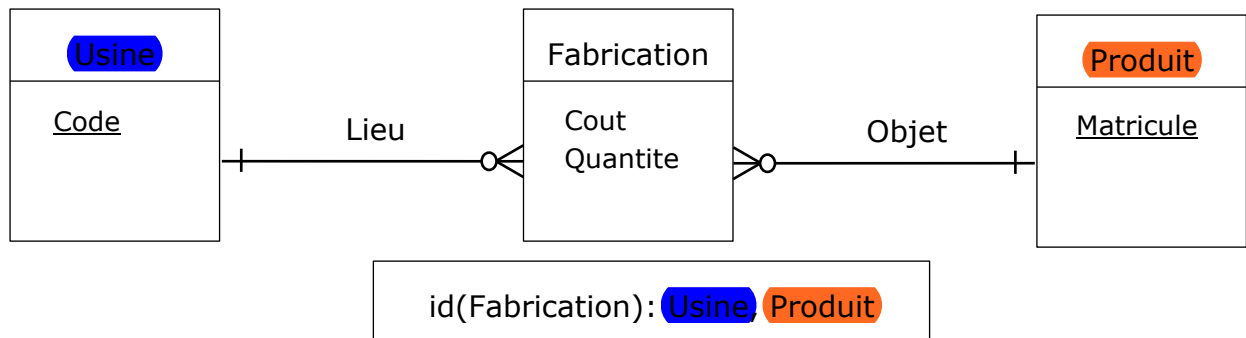
Soit l'association *Fabrication* entre les entités *Usine* et *Produit*



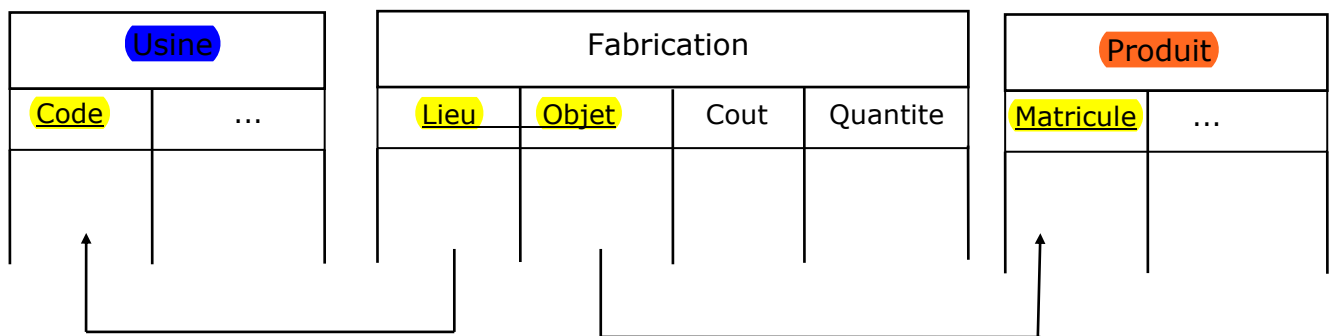
Ce schéma conceptuel doit d'abord être transformé.



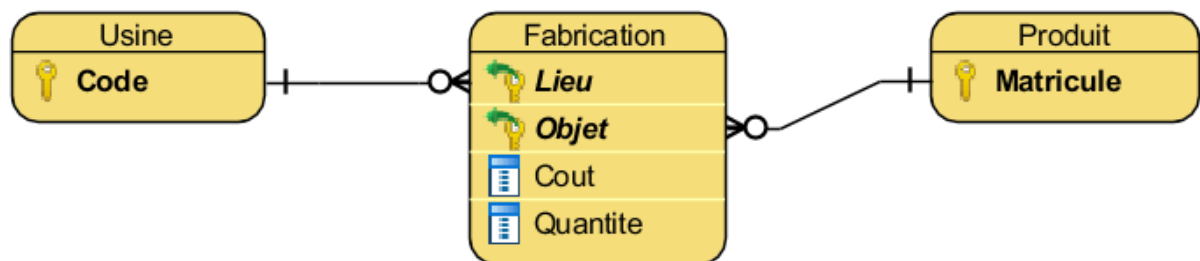
Des attributs peuvent être ajoutés à la table Fabrication, comme le coût unitaire et la quantité de produits fabriqués.



Le schéma conceptuel ainsi transformé est ensuite traduit en tables.



En Visual Paradigm :

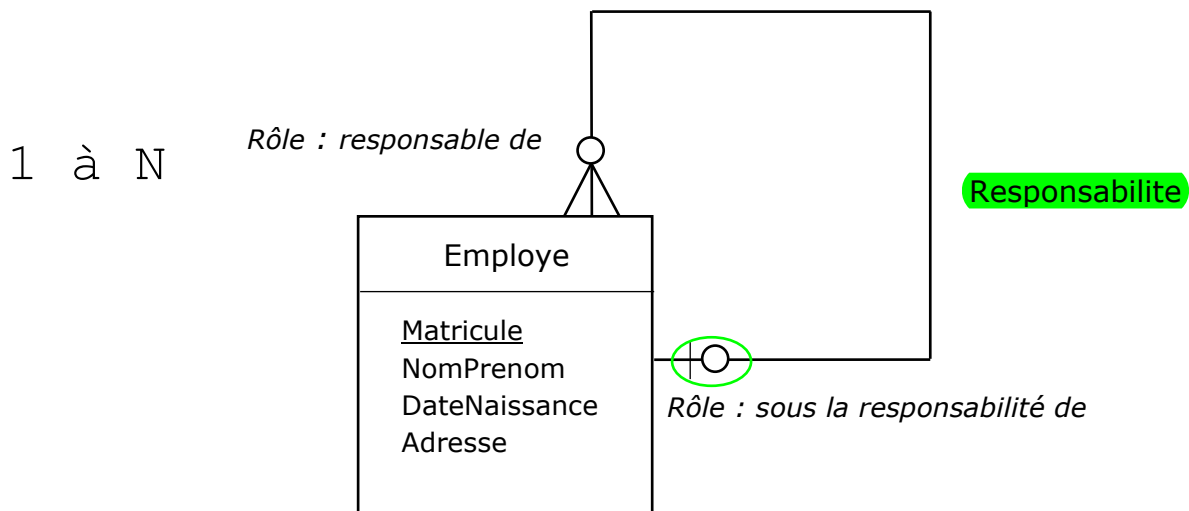


C. Représentation des associations cycliques

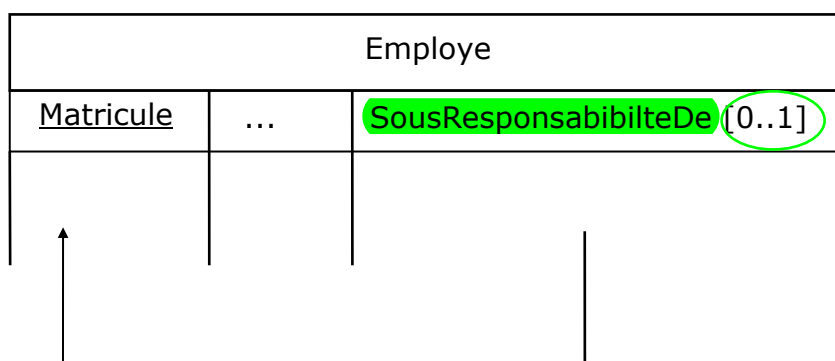
Une occurrence d'association **cyclique** est un lien entre **deux occurrences d'une même entité**. Les deux rôles d'une association cyclique sont donc définis sur la même entité.

Exemple 1

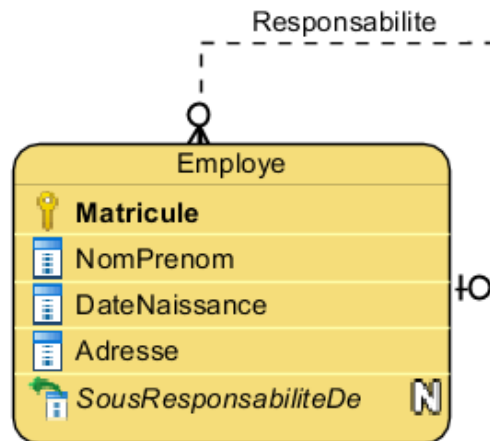
La hiérarchie dans une entreprise peut être représentée via l'association cyclique *Responsabilite*. Il s'agit d'une association 1 à N. Une clé étrangère suffit donc pour traduire cette association cyclique.



Puisque cette association *Responsabilite* traduit une relation 1 à N, il suffit d'ajouter une clé étrangère dans l'entité *Employe*, représentant le rôle ayant la cardinalité maximum = 1, à savoir, le rôle *sous la responsabilité de*.

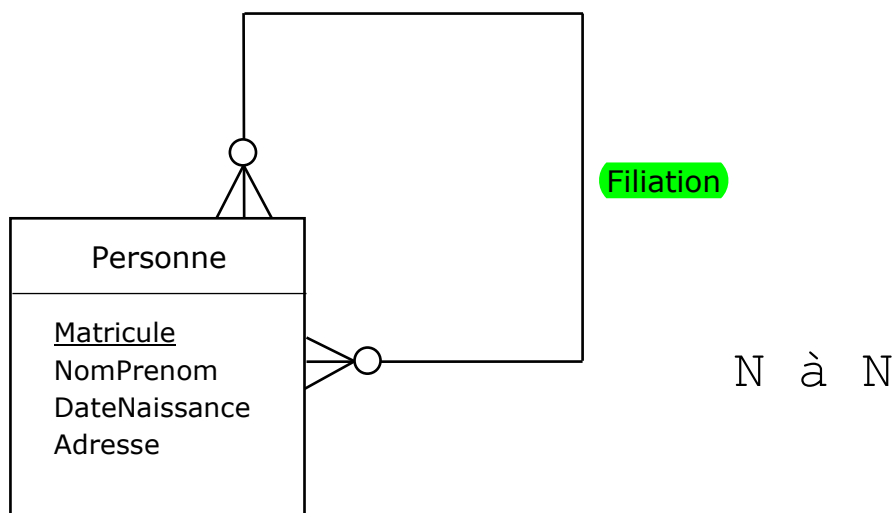


En Visual Paradigm :



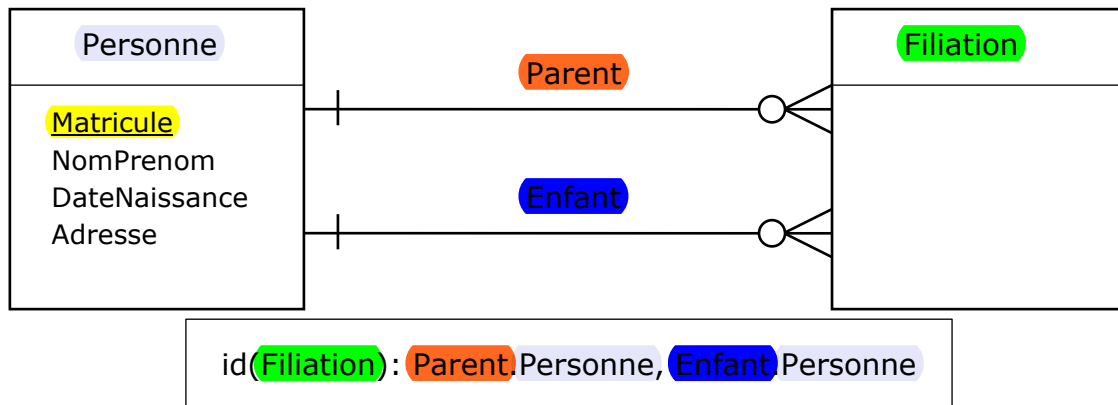
Exemple 2

Un autre exemple d'association cyclique est la relation de filiation. Une occurrence de filiation relie deux occurrences distinctes de l'entité *Personne* : une occurrence qui joue le rôle de l'enfant, et une occurrence qui joue le rôle du parent.



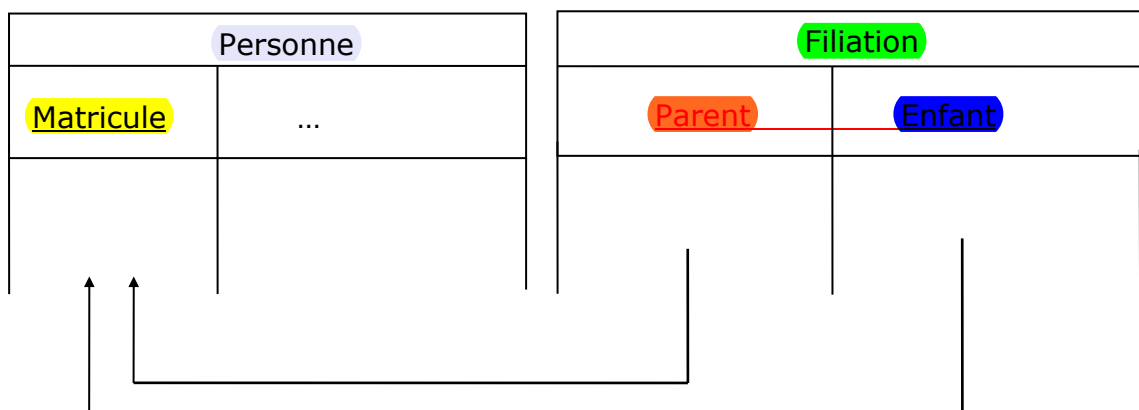
L'association *Filiation* représente une relation N à 2 (deux parents : un père et une mère) qui est un cas particulier d'une relation N à N. Pour le traduire en relationnel, il faut donc "éclater" l'association *Filiation* en deux associations 1 à N. La notion de filiation peut être vue comme concept plutôt que comme un lien.

Transformation en ...

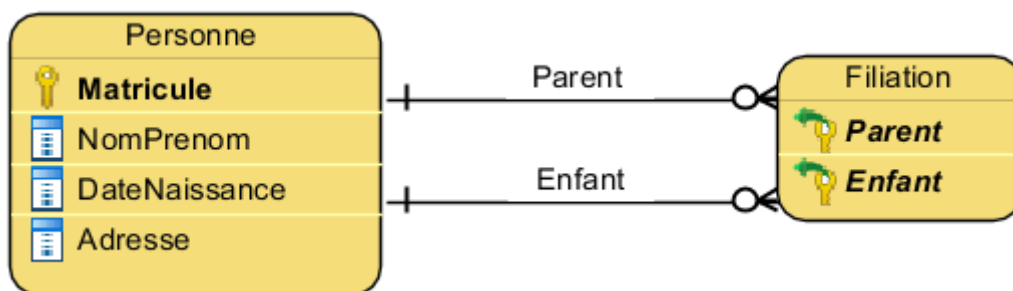


L'entité *Personne* sera représentée en relationnel par une table, la table *Personne*.

L'entité *Filiation* ainsi obtenue sera représentée par une nouvelle table ne contenant **que deux colonnes**, à savoir, les **deux clés étrangères** vers la table *Personne*.

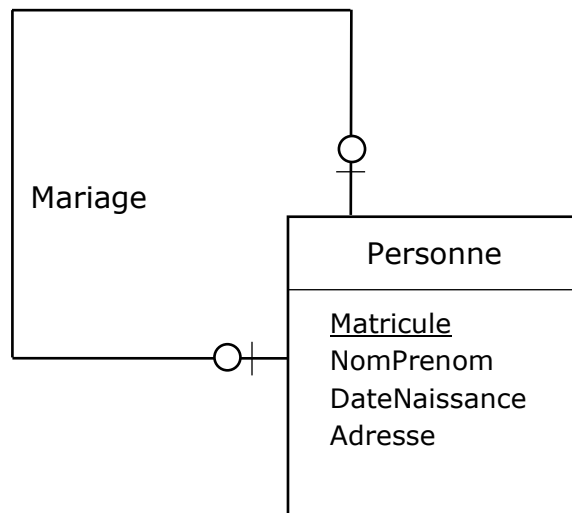


En Visual Paradigm :

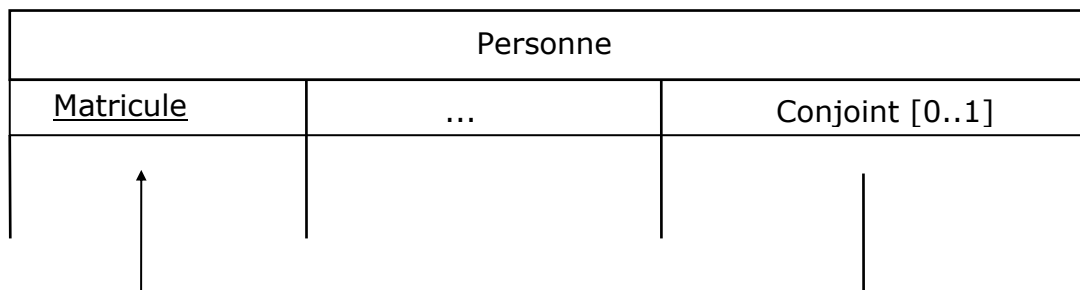


Exemple 3.

Les liens de mariage entre un homme et une femme peuvent être représentés via une association cyclique.

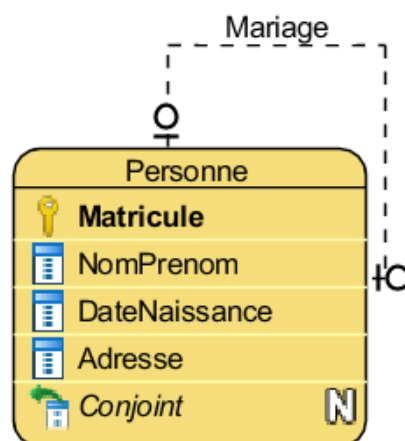


L'association *Mariage* traduit une relation 1 à 1. Il suffit d'ajouter une clé étrangère dans l'entité *Personne* représentant **un seul rôle**.



Attention, prévoir deux clés étrangères induirait une redondance.

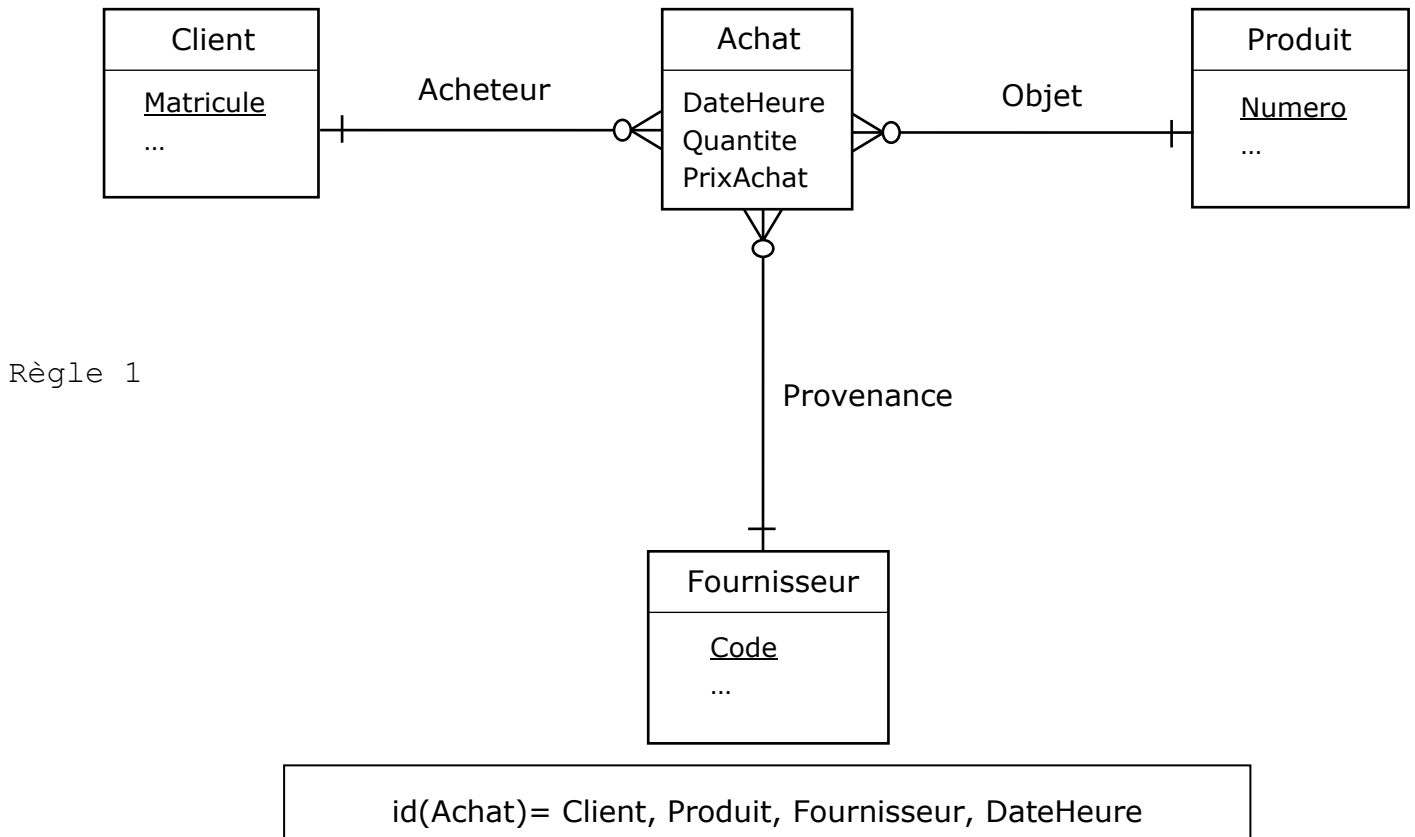
En Visual Paradigm :



° > 2

D. Représentation des relations de degré supérieur à 2

Soit la notion d'achat qui est un **lien**, une relation **entre des clients, des produits et des fournisseurs**. Il s'agit d'une relation **ternaire**. *Achat* est considéré comme un concept. Il fait donc l'objet d'une entité. Trois associations sont nécessaires : *Acheteur*, *Objet* et *Provenance*.



Les 4 entités *Client*, *Produit*, *Fournisseur* et *Achat* sont représentées en relationnel respectivement par les tables *Client*, *Produit*, *Fournisseur* et *Achat*. La table *Achat* contiendra, entre autres colonnes, trois clés étrangères vers les trois tables *Client*, *Produit* et *Fournisseur*.

3.5 Conclusion

Beaucoup de bases de données actuelles sont de type relationnel.

La maîtrise des bases de données relationnelles est donc vitale pour un futur informaticien.

C'est pourquoi, l'apprentissage du langage SQL est prévu au second quadrimestre du bloc 2. Au cours de ce laboratoire seront étudiés entre autres le langage relationnel de définition de données (RDDL) et le langage relationnel de manipulation de données (RDML).

4 Héritage et bases de données relationnelles

Cette partie du cours fera l'objet d'un support de cours qui sera accessible ultérieurement.

