

L'essentiel sur MERISE

Cours 3

Introduction

La méthode MERISE propose plusieurs modèles qui ont tous une fonctionnalité. Ce cours permet de faire une mise au point des cours précédents, tout en offrant de nouvelles perspectives.

Nous aborderons dans ce cours la classification des différents modèles existants ainsi que les liens qui les unissent.

Le problème de la normalisation sera abordé, ainsi que les extensions récemment proposées par la méthode.

Nous approcherons SQL pour ce qui est de la description physique des données. Pour finir nous verrons un petit florilège des erreurs à ne pas commettre.

1. Les différents modèles

Concernant les différents modèles, il faut bien avoir à l'esprit que ceux-ci dépendent avant tout de ce que l'on souhaite modéliser. Trois choix s'offrent à nous :

- Les données
- Les traitements
- Les flux

Remarquons que la maîtrise des modèles qui vont suivre n'est possible qu'avec une mise en pratique et une utilisation courante. Nous tâcherons de mettre en pratique chaque modèle, par contre il n'est pas évident que tous fassent l'objet d'une utilisation courante en entreprise.

Suivants les trois choix dégagés initialement, il faut prendre en compte le niveau de modélisation. Là encore, plusieurs choix s'offrent à nous :


- Niveau conceptuel
- Niveau organisationnel
- Niveau logique
- Niveau physique

Remarque : Nous tenterons de traiter de manière pratique les différentes possibilités offertes par MERISE. Pour cela, plusieurs TD accompagneront ce cours. Ayons tout de même à l'esprit que vouloir traiter MERISE dans sa globalité reste prétentieux (de ma part).

2. La classification

L'état de l'art, nous permet de faire la proposition suivante :

niveau	données	traitements	flux
conceptuel	MCD	MCT	MCF
organisationnel	MOD	MOT	MOF
logique	MRD	DLT	-
physique	DPD	-	-



Les flux représentent une extension des traitements.

Abréviations :

MCD : Modèle Conceptuel de Données

Il adopte le formalisme que vous connaissez déjà (modèle entité-relation)

MOD : Modèle Organisationnel de Données

Il adopte la même représentation que le MCD, il tient simplement compte de l'organisation du système à modéliser.

Les données (du MCD) ne faisant pas l'objet de traitements automatisés ne seront pas retenues (dans le MOD).

L'identification des types de site (siège, agence, ...) et des types d'acteur (service financier, client, comptable, ...) est nécessaire.

Il peut permettre de définir les droits (consultation ou CMA) d'accès aux objets (tables) suivants les types d'acteur.

MRD : Modèle Relationnel de Données :

Il adopte la représentation du schéma relationnel. Il semble présomptueux (d'après plusieurs ouvrages) de présenter un modèle pour le niveau logique (adieu MLD !). Le niveau logique englobe trop de composants techniques qui ont une influence sur lui (middleware, SGBDR, outils de développement, serveur de données et de traitements).

DPD : Description Physique des Données

Cette description se fait via le langage SQL (Structured Query Language), langage normalisé utilisé sur les différents SGBDR sauf ACCESS¹. Cette description utilise le Langage de Définition de Données (LDD) de SQL. Le niveau physique est lui aussi débarrassé d'une représentation sous forme de modèle (même problème que pour le niveau logique)(adieu...MPD !).

MCT : Modèle Conceptuel de Traitement

Sa représentation utilise un formalisme bien défini. Tout système renferme au moins une activité. Cette activité est décomposée en processus. Un processus se décompose en opérations.

MOT : Modèle Organisationnel des traitements

Le formalisme est différent du MCT. Il complète le MCT en prenant en considération l'organisation de l'entreprise :

- Affectation aux postes de travail (locaux, centraux)
- Niveau et le type d'automatisation des traitements (manuel, temps réel, temps différé)

Le MOT permet de décrire des phases. Une opération du MCT correspondra à une ou plusieurs phases dans le MOT.

DLT : Description Logique des Traitements

La DLT n'offre pas de formalisme normalisé. En effet ce niveau est non stabilisé, en raison de l'évolution régulière et rapide des techniques. Cette DLT est basée sur les phases du MOT. On peut classer chaque phases en phase transactionnelle ou en phase batch. Une prise en compte des couches de traitement est nécessaire :

¹ Remarque : Tous les SGBDR utilisent un dialecte SQL, dérivé du SQL original issu de SEQUEL. ACCESS a le bon goût de s'illustrer par un dialecte plus exotique comparé aux autres produits.

- **Présentation** : présentation des données vis à vis des utilisateurs. Cette partie des traitements comprend l'ensemble des tâches de préparation d'affichage des données, de contrôle de premier niveau ne nécessitant pas l'accès à la base de données.
- **Traitements** : tâches décrivant les traitements de contrôle, calcul et préparation de mise à jour ou d'édition.
- **Données** : accès à la base de données. Tous les traitements permettant d'accéder aux données qui incluent le parcours du modèle de données et les traitements dédiés aux erreurs d'accès.

MCF : Modèle Conceptuel des Flux

Le modèle conceptuel des flux est constitué de sous-modèles :

- MC : Modèle de contexte
- MD : Modèle Détaillé

Le modèle de contexte représente le contexte du système. Il représente l'extérieur du système (son environnement). C'est une boîte noire.

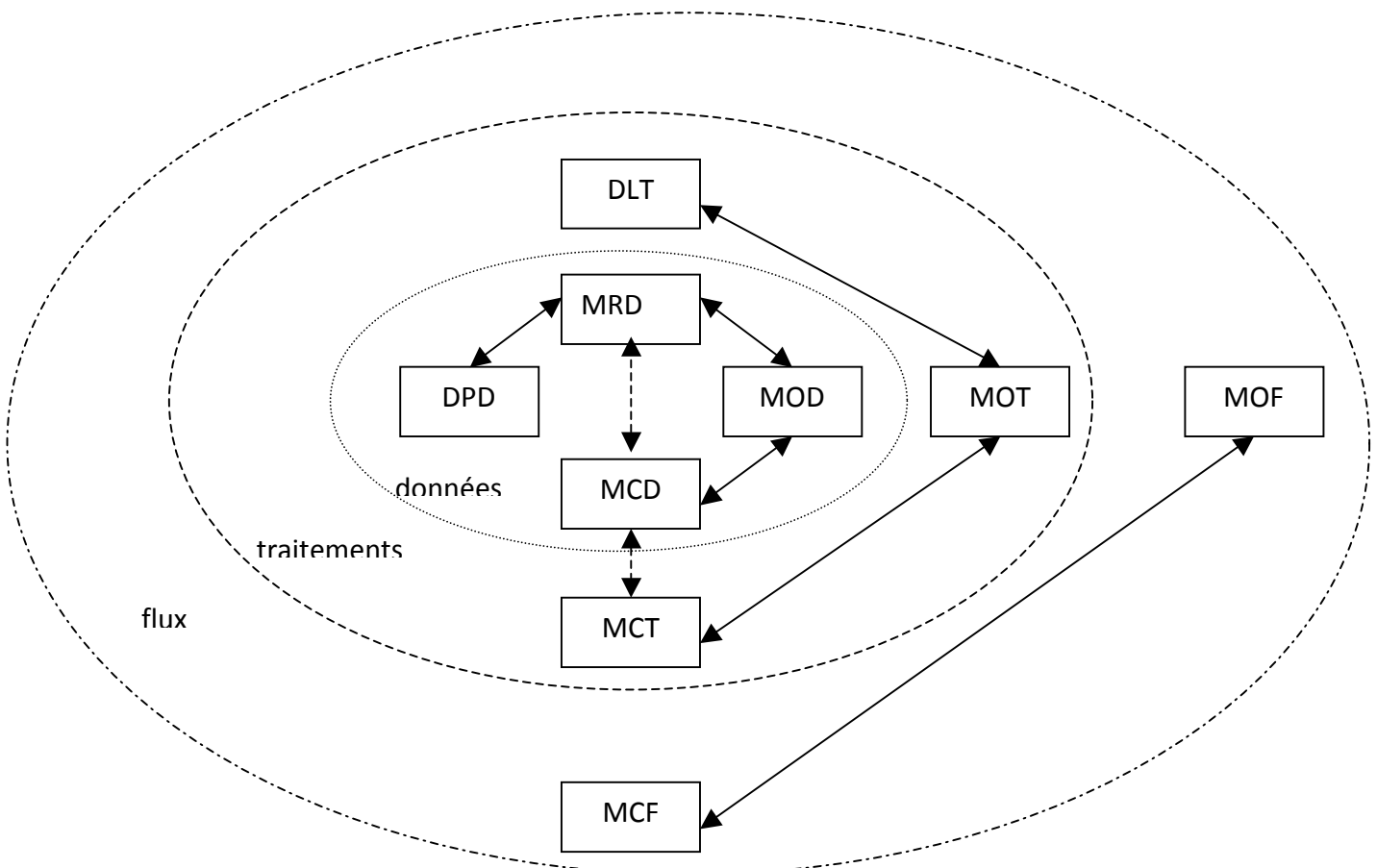
Le modèle détaillé fait la lumière sur la boîte noire. Il enregistre les activités et non les acteurs. La concaténation du MC et du MD donne naissance au MCF.

MOF : Modèle Organisationnel des Flux

Ce modèle utilise des flux et des acteurs. On passe de la notion d'activité à la notion d'acteurs ou de services. Ce niveau organisationnel peut renfermer une matrice des flux.

3. Les liens entre les modèles

Le schéma présenté ici formalise les liens possibles entre les différents niveaux.



4. La normalisation

Un modèle conceptuel de données doit être validé avec un ensemble de règles. Ces règles correspondent au processus de **normalisation**.

En règle générale on considère qu'un MCD est normalisé lorsqu'il est en **3 FN**. Cela signifie qu'il existe des niveaux de **normalisation** :

- 1 FN : **première forme normale**
- 2 FN : **deuxième forme normale**
- 3 FN : **troisième forme normale** (MCD normalisé)
- 4 FN : **quatrième forme normale** (non traité dans ce cours)
- 5 FN : **cinquième forme normale** (non traité dans ce cours)

Pour atteindre la troisième forme normale, il y a **deux manières** de procéder :

- Soit on **passe directement** le modèle en troisième forme normale grâce à un ensemble de règles que l'on applique aux entités et aux relations.
- Soit on **passe le modèle** en 1FN, puis en 2FN puis en 3FN, en appliquant à chaque fois la ou les règles qui correspondent au **niveau** de normalisation traité. On travaille là encore au niveau des entités et des relations.

4.1. Passage direct en 3 FN (le plus simple)

Ce passage s'accompagne de l'application des **6 règles** suivantes :

Règles concernant les propriétés :

- Règle n°1 : toutes les propriétés doivent être élémentaires, c'est à dire non décomposables. Nous touchons ici, la notion d'atomicité des propriétés. Il faut en fait considérer une entité comme une molécule. La propriété étant l'atome constituant cette molécule. Une molécule étant constituée d'atomes, une entité est constituée de propriétés. Ayons bien à l'esprit que l'atome ne prend qu'une seule valeur (comme dans la classification chimique de Mendeleïev).

Règles concernant les entités :

- Règle n°2 : Existence d'un identifiant pour chaque entité.
- Règle n°3 : Toutes les propriétés autres que l'identifiant doivent être en dépendance fonctionnelle complète et directe de l'identifiant. Par conséquent pour chaque occurrence d'une entité, chaque propriété doit prendre une et une seule valeur (il ne peut donc y avoir ni valeurs répétitives, ni absences de valeurs pour une même propriété).

Règles concernant les relations :

- Règle n°4 : Toutes les propriétés d'une relation doivent dépendre complètement de l'identifiant de la relation ; de plus, chaque propriété doit dépendre de tout l'identifiant et non d'une partie de cet identifiant. Ainsi pour chaque occurrence d'une relation, chaque propriété doit être en dépendance fonctionnelle de l'identifiant de la relation et doit donc prendre une et une seule valeur.

Règles concernant l'ensemble du MCD :

- Règle n°5 : Une propriété ne peut apparaître qu'une seule fois dans un même MCD. Elle ne peut qualifier qu'une seule entité ou une relation.

- Règle n°6 : Les propriétés qui sont le résultat d'un calcul ne doivent pas, en principe, figurer dans un MCD sauf si elles sont indispensables à la compréhension de celui-ci.

4.2. Passage non direct en 3FN (le plus compliqué)

Ce mode de transformation sera apprécié dans le cas où l'on souhaite connaître **l'état du modèle à chaque niveau intermédiaire**.

Là encore cette normalisation s'applique sur les entités et les relations.

✎ Nous ne donnerons ici que la description sommaire des trois premières formes normales, la normalisation étant traitée en détail dans les ouvrages spécialisés en bases de données.

- **1FN : DF, attribut atomique :**

$A \rightarrow^2 B$, A et B prennent une seule valeur
En 1 FN DF, et atomicité.

- **2FN : DF élémentaire** (cas des clés multi-attributs)

DF élémentaire : Une DF $A1 \rightarrow A2$ est élémentaire s'il n'existe pas un attribut A3 contenu dans A1 tel que : $A3 \rightarrow A2$. C'est le problème posé par les identifiants composés ou multi-attributs

Soit A1 identifiant clé composée (numéro facture, numéro produit)

Soit la quantité

$A1 \rightarrow A2$

Soit A3 numéro facture ou numéro produit

~~$A3 \rightarrow A2$~~

- **3FN : DF directe** (non transitive)

DF directe : Une DF élémentaire $A1 \rightarrow A2$ est directe s'il n'existe pas A3 tel que $A1 \rightarrow A3$ et $A3 \rightarrow A2$ ($A1 \rightarrow A3 \rightarrow A2$)

5. Les extensions

Apparues avec MERISE/2 elles offrent de nouvelles possibilités en matière de modélisation.

Mais avant cela quelques définitions.

5.1. Les dépendances fonctionnelles

Le dictionnaire de données contient toutes les propriétés utiles à notre système d'information.

Il nous reste à créer des liens entre ces propriétés afin d'exprimer la réalité.

Définition : Deux propriétés sont en **dépendance fonctionnelle** si la connaissance **d'une valeur** de la **première** permet de déterminer la connaissance **d'une valeur** de la **seconde**. La première propriété est dite **SOURCE** la deuxième **BUT**.

Formalisme : Une dépendance fonctionnelle est notée par une flèche allant de la propriété source vers la propriété but.

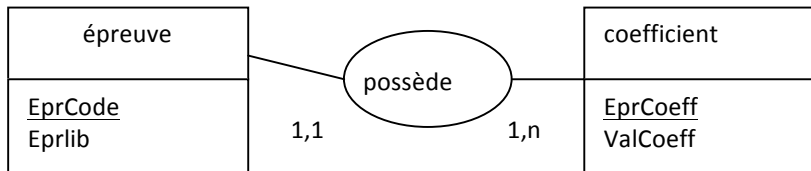
² DF : Dépendance Fonctionnelle symbolisée par \rightarrow

Exemple : "Toute épreuve de l'examen possède un coefficient et un seul "



Analyse :

Voyons de plus près le MCD correspondant :



Le MRD aura la forme suivante :

épreuve(EprCode, EprLib, #EprCoeff)
 coefficient(EprCoeff, ValCoeff)

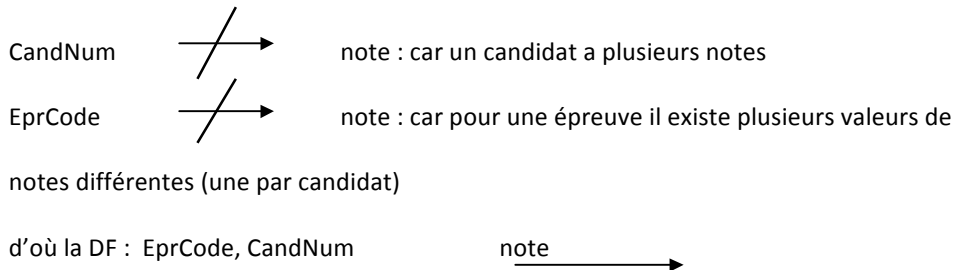
Conclusion :

Une valeur de EprCode définit une valeur de EprLib et une valeur de EprCoeff.

Remarque :

Les DF n'ont pas toujours comme source une seule propriété. Pour mettre en évidence ces DF, il faut repérer les propriétés pour lesquelles il n'a pas été possible de trouver une DF dont la source soit composée d'une seule propriété.

Exemple : La propriété NOTE



La source d'une dépendance fonctionnelle constitue une CLE ou un INDEX ou IDENTIFIANT. Une clé ne doit pas subir de mise à jour = stabilité.

5.2.1. Propriétés des dépendances fonctionnelles

- **la réflexivité** : $A \rightarrow A$ (A propriété quelconque est en DF sur elle même)
- **L'augmentation** : si $A \rightarrow B$ alors $A, C \rightarrow B \quad \forall C$ (quelque soit C)
- **La transitivité** : si $A \rightarrow B$ et $B \rightarrow C$ alors $A \rightarrow C$
- **La pseudo-transitivité** : Si $A \rightarrow B$ et $B, C \rightarrow D$ alors $A, C \rightarrow D$
- **L'union** : si $A \rightarrow B$ et $A \rightarrow C$ alors $A \rightarrow B, C$
- **La décomposition** : si $A \rightarrow B, C$ alors $A \rightarrow B$ et $A \rightarrow C$

Exemple montrant l'intérêt de l'application de ces propriétés remarquables :

Considérons la gestion des locations dans une agence immobilière. Une partie de l'analyse est traduite par les trois dépendances fonctionnelles suivantes :

$\text{Id_Appartement} \rightarrow \text{Id_Propriétaire}$
 $\text{Id_Appartement} \rightarrow \text{Id_Locataire}$
 $\text{Id_Appartement, Id_Propriétaire, Id_Locataire} \rightarrow \text{Id_Contrat}$

La dernière dépendance traduit la perception que l'analyste a eu du contrat, à savoir que celui-ci est identifié par un triplet constitué d'un appartement, d'un propriétaire et d'un locataire.

On déduit de ce jeu de dépendances :

$\text{Id_Appartement} \rightarrow \text{Id_Propriétaire}$
 $\text{Id_Appartement} \rightarrow \text{Id_Locataire}$
 $\text{Id_Appartement} \rightarrow \text{Id_Appartement}$ (réflexivité)

De ces 3 dépendances, en appliquant la propriété d'union, on obtient la nouvelle dépendance :

$\text{Id_Appartement} \rightarrow \text{Id_Appartement, Id_Propriétaire, Id_Locataire}$

Comme,

$\text{Id_Appartement, Id_Propriétaire, Id_Locataire} \rightarrow \text{Id_Contrat}$

On peut en déduire, en application de la propriété de transitivité que

$\text{Id_Appartement} \rightarrow \text{Id_Contrat}$

Ainsi, nous avons simplifié les dépendances fonctionnelles décrivant ce domaine aux 3 dépendances élémentaires suivantes :

$\text{Id_Appartement} \rightarrow \text{Id_Propriétaire}$
 $\text{Id_Appartement} \rightarrow \text{Id_Locataire}$
 $\text{Id_Appartement} \rightarrow \text{Id_Contrat}$

5.2.2. Typologie des dépendances fonctionnelles

DF élémentaire : Une DF $A1 \rightarrow A2$ est élémentaire s'il n'existe pas un attribut $A3$ contenu dans $A1$ tel que : $A3 \rightarrow A2$. C'est le problème posé par les identifiants composés ou multi-attributs.

Ainsi il ne faut pas d'attributs superflus dans la partie gauche de la dépendance

Exemple : 1°) $\text{CmdNum, ArtRef} \rightarrow \text{QtéCmdée}$
 2°) $\text{CmdNum, ArtRef} \rightarrow \text{ArtLib}$
 3°) $\text{ArtRef} \rightarrow \text{ArtLib}$

La 2°) est une DF non élémentaire car 3°) existe

DF directe : Une DF élémentaire $A1 \rightarrow A2$ est directe s'il n'existe pas $A3$ tel que $A1 \rightarrow A3$ et $A3 \rightarrow A2$

🔗 **Une DF directe est une DF qui n'est pas transitive.**

Exemple : 1°) $\text{FactNum} \rightarrow \text{RepNum}$ (N°représentant)
 2°) $\text{FactNum} \rightarrow \text{RepNom}$

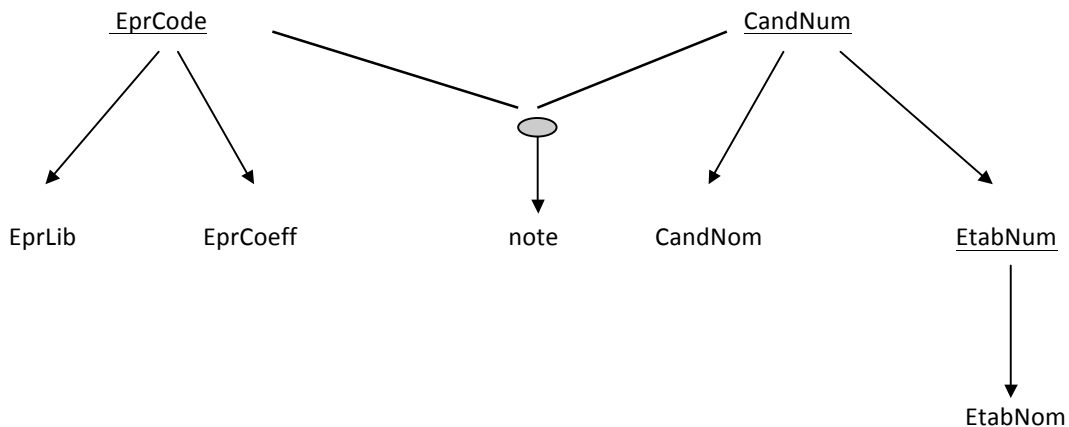
- 3°) FactNum \rightarrow FactDate
 4°) RepNum \rightarrow RepNom

La 2°) DF n'est pas directe car il existe 1°) et 4°).

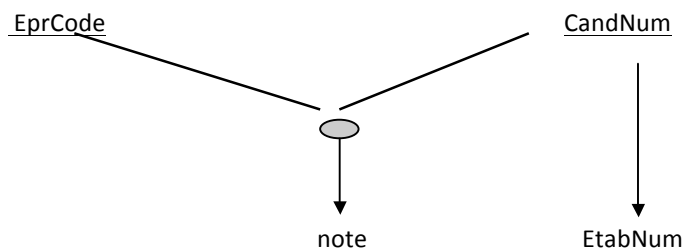
5.2.3. Graphe des dépendances fonctionnelles

Il **permet de vérifier** que les **propriétés** de notre système d'information sont toutes **reliées** par des **dépendances fonctionnelles** élémentaires et directes.

Le graphe permet aussi de mettre en évidence les DF transitives (non directes).

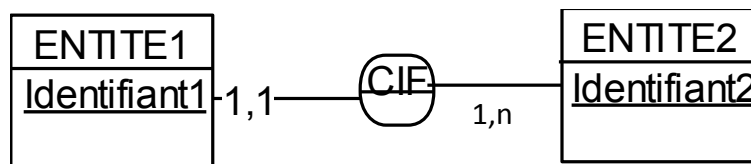


On peut remplacer ce graphe des DF par le graphe des clés

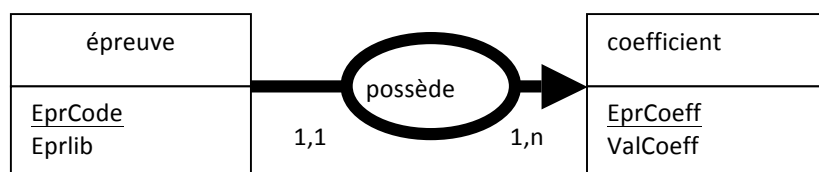


5.3. Les contraintes d'intégrité fonctionnelle

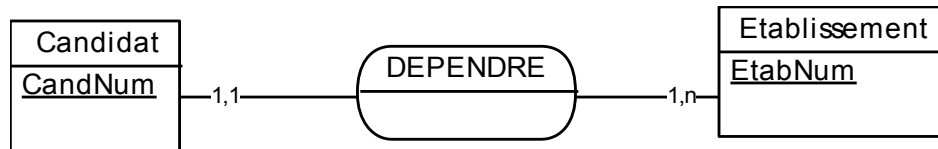
La **dépendance fonctionnelle entre identifiants de deux entités** est représentée dans le modèle de données par une **CONTRAINTE D'INTEGRITE FONCTIONNELLE (CIF)** ce qui implique qu'il n'est pas possible de créer une occurrence dans l'entité entité1, s'il n'existe pas au préalable une occurrence associée dans l'entité 2.



La meilleure représentation (d'une CIF) qui n'altère pas la représentation sémantique du schéma est la suivante :



Autre exemple de CIF : un candidat (CandNum) appartient à un et un seul établissement (EtabNum). On ne peut pas créer une occurrence dans l'entité CANDIDAT si on ne sait pas à quel ETABLISSEMENT il appartient.



Dans notre dernier cas, nous ne précisons pas que nous sommes en présence d'une CIF.

🔧 **Nous renoncerons** à utiliser les diverses notations en vigueur. Celles-ci sont précisées dans le cas où vous seriez confronté à ce genre de notation. Vous pourrez ainsi les comprendre.

5.4. Extensions MERISE/2

Pourquoi avoir introduit des extensions au modèle de données standard ?

Pour plusieurs raisons :

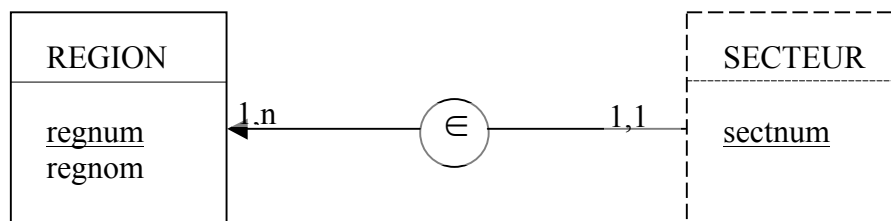
- Résoudre les problèmes qui ne l'étaient pas avec MERISE.
- Préciser les objets en faisant ressortir des propriétés, des contraintes d'intégrité et des notions d'historique.
- Aider à la construction du modèle des traitements et à sa validation.

A. Entité forte/entité faible

Dans la définition, une entité a une existence propre. Mais dans la réalité, il existe, dans certains cas des entités dont l'existence même est liée à d'autres entités.

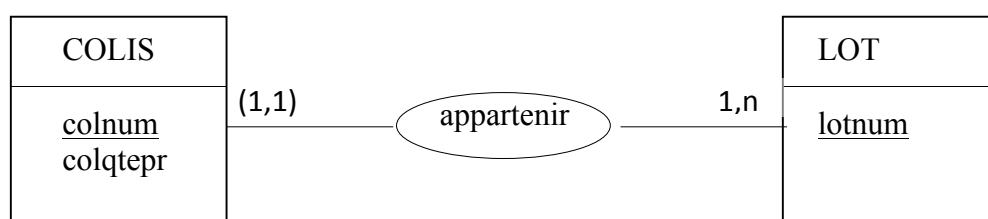
Prenons par exemple le cas d'un secteur qui est propre à une région, le numéro de secteur n'est pas un identifiant à proprement parler, il dépend du numéro de région. On parlera pour l'entité secteur **d'entité dépendante**, son identification étant une **identification relative** : la propriété sectnum est un numéro d'ordre 1,2 ...

Plusieurs notations sont proposées :



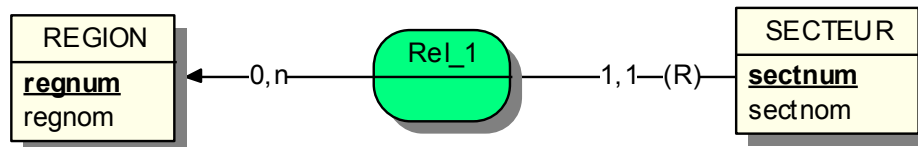
Ici **l'entité faible (ou dépendante)** est représentée en pointillé. Son lien de dépendance est indiqué par la flèche vers l'entité région. On notera que les cardinalités 1,1 représentées ne signifient pas une dépendance fonctionnelle, car le numéro d'ordre du secteur définit en fait plusieurs régions.

Une autre représentation (c'est le formalisme adopté par AMC*DESIGNOR):



Dans ce cas l'entité faible est l'entité COLIS, son identifiant est relatif au lot qui le contient. Les parenthèses autour des cardinalités 1,1 mettent en évidence le lien de dépendance d'une part, et qu'il ne s'agit pas d'une dépendance fonctionnelle d'autre part.

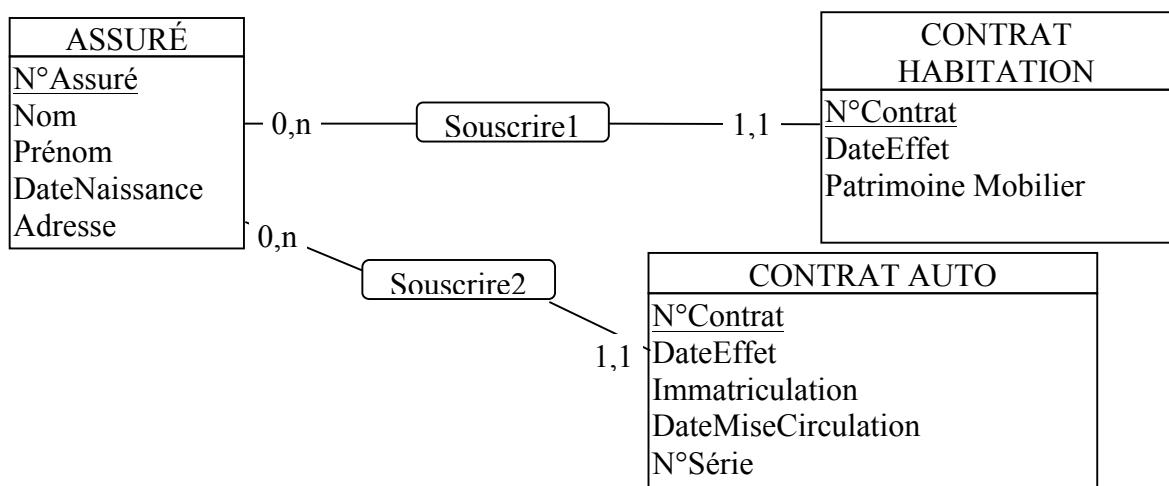
Enfin on trouve aussi le (R) de Relatif (formalisme Win'DESIGN) :



B. Généralisation/spécialisation

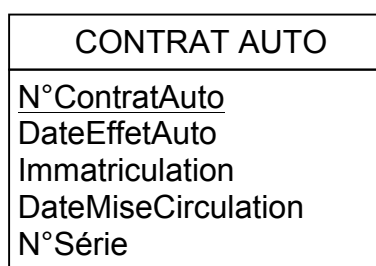
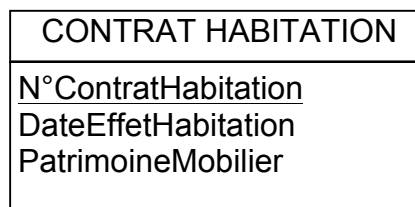
Soit le MCD suivant :

Modèle conceptuel de données (proposé et faux !)



B.1. Spécialisation, sans généralisation

Deux entités avec répartition de propriétés :



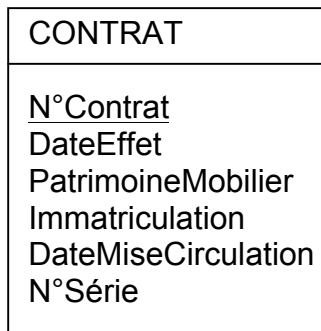
Les contrats habitations et les contrats auto sont décrits par un certain nombre de propriétés identiques communes à tous les contrats.

Problème ?

Redondance de propriétés qui pourraient être regroupées

B.2. Généralisation sans spécialisation

Une seule entité avec toutes les propriétés



- ✓ Pour chaque occurrence, toutes les propriétés n'ont pas toutes un sens. Ainsi pour un contrat auto les propriétés relatives aux contrats habitation n'ont pas de sens.

Problème ?

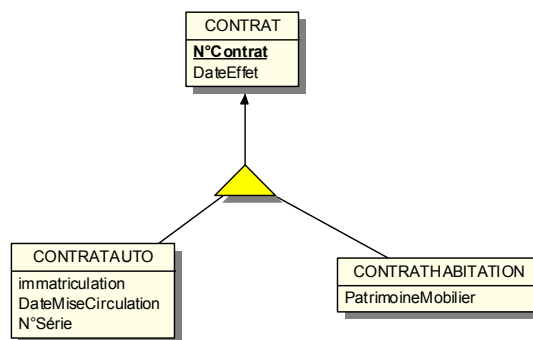
Des propriétés ne seront pas renseignées, donc inutiles pour certaines occurrences.

- ✓ Rien ne permet de distinguer facilement les contrats autos des contrats habitations.
- ✓ Pour certains n-uplets, des attributs auront un sens, pour d'autres pas, et ce en fonction du sous-types d'appartenance : c'est le phénomène d'expropriation de propriété, qui oblige au niveau de la table générique d'autoriser la valeur nulle pour certains attributs.

B.3. Solution : Concept de généralisation/spécialisation

Ce concept est fondé sur le fait de regrouper toutes les propriétés communes de plusieurs entités de base dans une entité dite **générique ou type**, et de créer plusieurs entités **spécialisées ou sous-types** ne contenant que les propriétés spécifiques de chaque entité de base.

Ces deux sous-types d'entités étant liés par un **lien de généralisation/spécialisation** à l'entité générique.



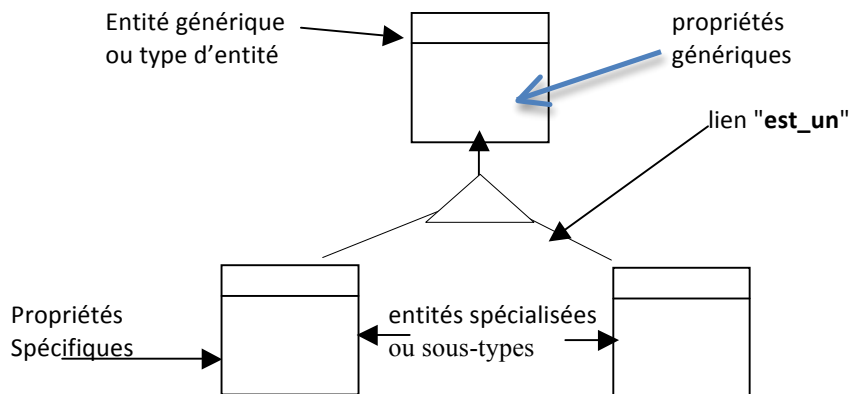
Cette représentation est plus proche de la réalité, elle corrige en partie le MCD proposé initialement.

B.4. Définition de la généralisation/spécialisation

La **généralisation/spécialisation** est un **processus** de modélisation permettant de **lier** une **entité générique** (ou sur-type) et une ou plusieurs **entités spécialisées** (ou sous-types). **L'entité générique contient** les **propriétés communes** à tous les **sous-types**. Les entités spécialisées **héritent** des propriétés du sur-type et de plus, définissent des propriétés qui leurs sont **spécifiques**.

Ainsi dans l'exemple ci-dessus, pour chaque occurrence d'un contrat automobile on détermine le numéro du contrat, date d'effet, immatriculation du véhicule, Date mise en circulation et numéro de série.

① **La généralisation/spécialisation simple** est caractérisée par l'unicité du lien de généralisation spécialisation pour une même entité



Le lien "est-un" qui indique que **toutes** les occurrences de l'entité spécialisée sont aussi occurrences de l'entité générique et **certaines** occurrences de l'entité générique sont aussi occurrences de l'entité spécialisée.

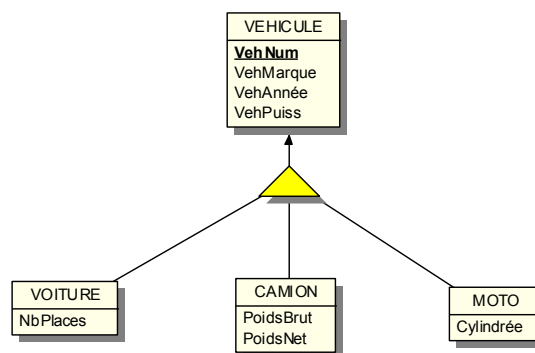
Prenons notre exemple : L'entité générique CONTRAT et les deux entités spécialisées CONTRAT AUTO et CONTRAT HABITATION.

Les n°contrat, date d'effet des contrats auto sont **tous** dans l'entité générique contrat.

Les n°contrat, date d'effet des contrats habitation sont **tous** dans l'entité générique contrat.

Les n°contrat et date d'effet des contrats de l'entité générique sont relatifs soit à des contrats auto, soit à des contrats habitation, d'où le terme "**certaines** occurrences".

EXEMPLE de spécialisation avec trois sous-type :



Complément : Il peut y avoir transfert d'une occurrence d'une entité spécialisée vers une autre entité spécialisée autrement dit, d'un sous-type vers un autre. On parle dans ce cas de **transmutation**.

Par exemple, si on considère l'entité générique PERSONNEL et les entités spécialisées VACATAIRES d'une part et MENSUEL d'autre part les employés VACATAIRES peuvent devenir MENSUEL. Ils passent d'un sous-type à l'autre, on parle dans ce cas de transmutation.

Dans l'exemple ci-dessus cela n'est pas possible pour un sous-type de passer d'une spécialisation à une autre.

Les règles d'**héritages** diffèrent suivant les traitements, en règle générale dans le cas d'une spécialisation simple : les entités spécialisées héritent de toutes les propriétés de l'entité générique. On peut aussi l'exprimer autrement : les sous-types « reçoivent » toutes les propriétés de l'entité générique.

Dans l'exemple ci-dessus la relation déduite de l'entité spécialisée VOITURE comprend les propriétés (vehnum, vehmarque, vehannée, vehpuiss, nbplaces).

La relation déduite de l'entité spécialisée CAMION comprend les propriétés (vehnum, vehmarque, vehannée, vehpuiss, poidsbrut, poidsnet).

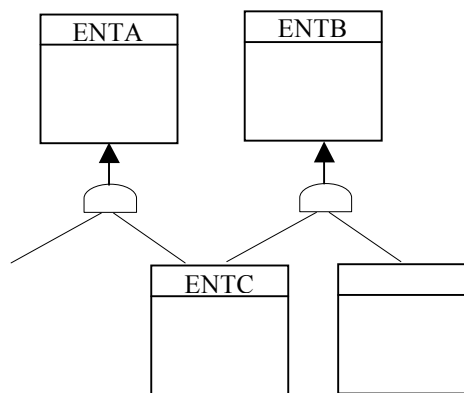
La relation déduite de l'entité spécialisée MOTO comprend les propriétés (vehnum, vehmarque, vehannée, vehpuiss, cylindrée).

Il n'existe pas forcément de relation véhicule (c'est le choix du concepteur de garder ou non cette relation).

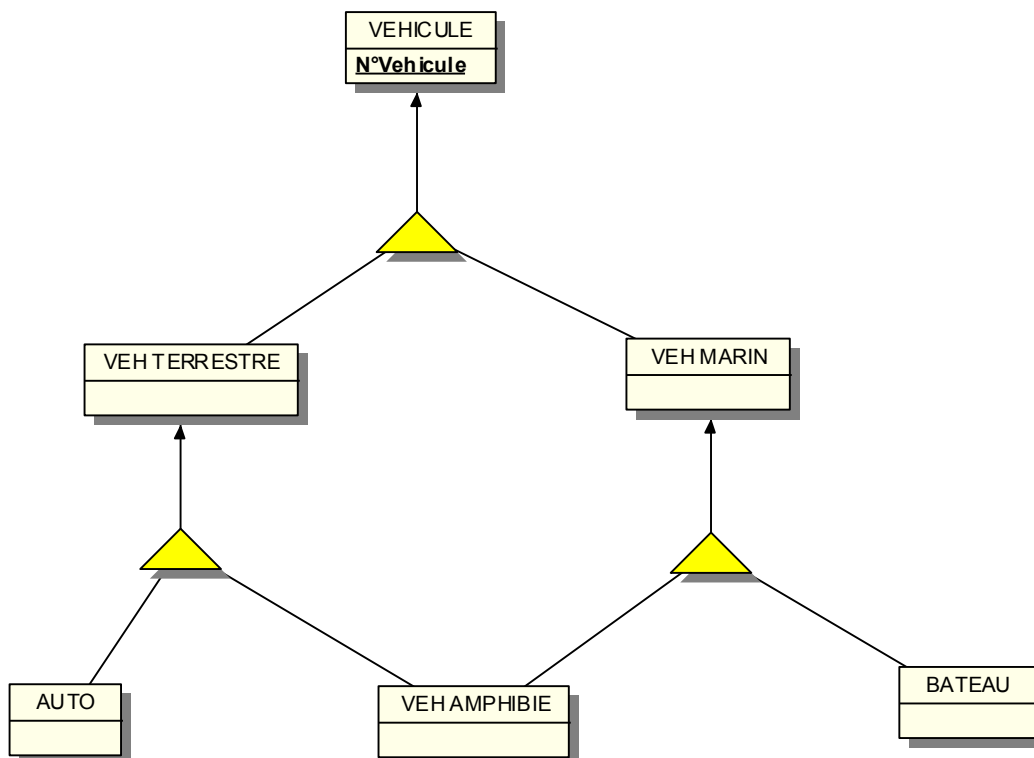
② La généralisation/spécialisation multiple

Dans la plupart des cas, les entités spécialisées sont situées dans une hiérarchie de classe simple. Toutefois il existe des cas où les entités spécialisées sont situées dans plusieurs classes d'héritage : on parle alors de généralisations multiples.

Dans notre exemple, l'entité C est dans la hiérarchie de classe de l'entité A ET dans la hiérarchie de classe de l'entité B



Regardons l'exemple suivant :



C. Contraintes sur spécialisation

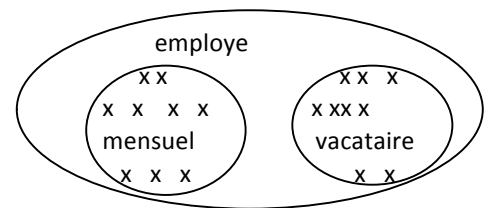
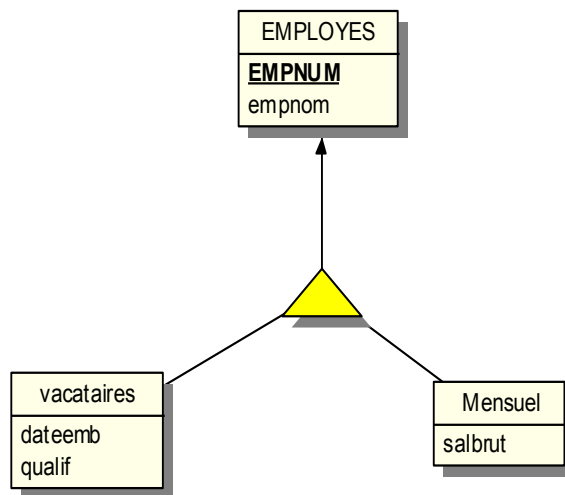
C.1 - Les contraintes de base

Les deux contraintes de base qui sont utilisées pour introduire les contraintes d'intégrité dans les modèles de données avec MERISE/2 sont la couverture et la disjonction.

C.1.1. La couverture

Cette contrainte définit si toutes les occurrences d'une entité générique sont représentées par les entités spécialisées liées à cette entité générique. On parle de non couverture dans le cas contraire.

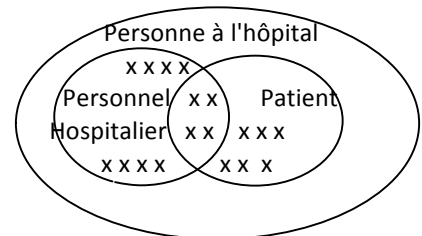
Exemple de couverture :



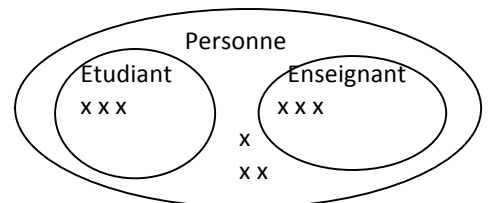
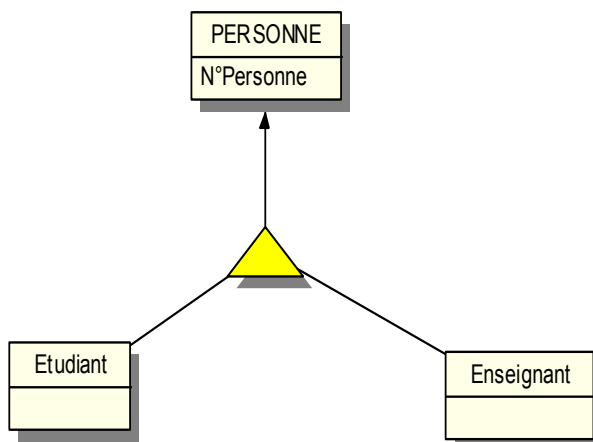
Les employés sont soit mensuels, soit vacataires

Autre exemple de couverture :

Une personne à l'hôpital fait partie du personnel hospitalier ou est un patient et parfois fait partie des deux spécialisations.



Contre exemple : Non-couverture

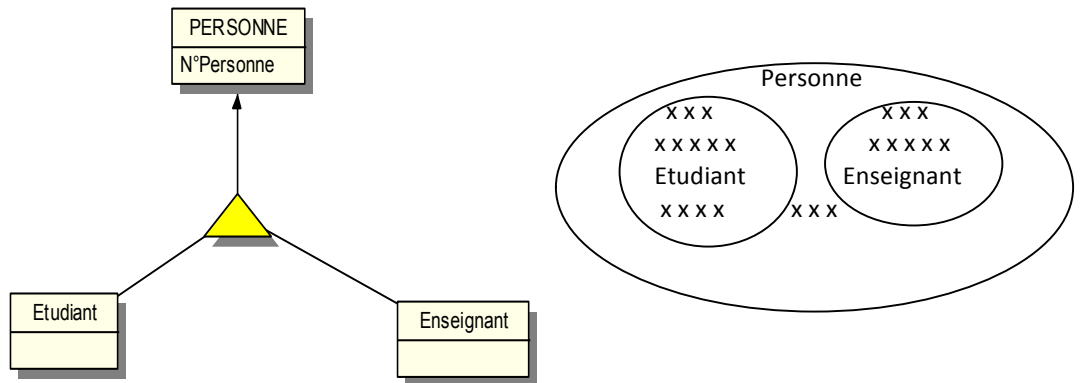


Dans un établissement scolaire, il peut y avoir des personnes qui ne sont pas enseignants ni étudiants

C.1.2 - La disjonction

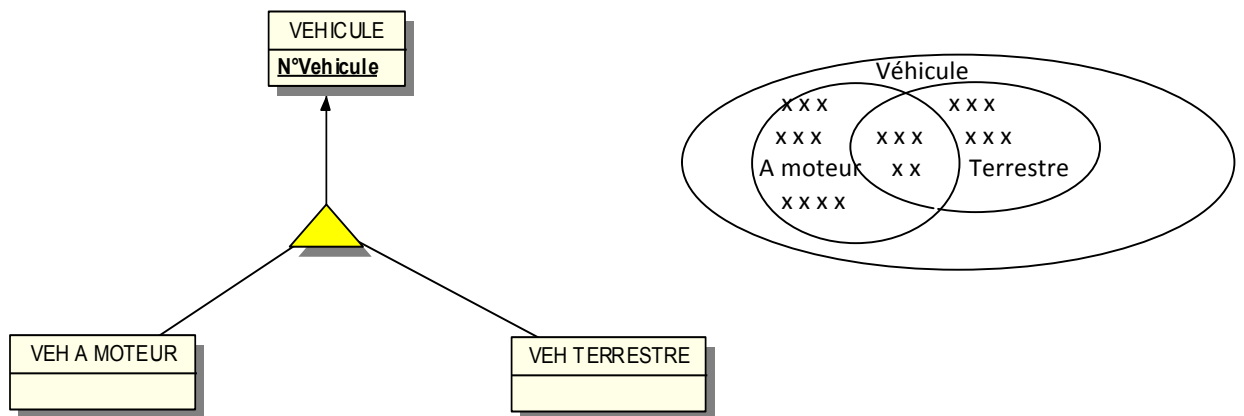
Cette contrainte permet de savoir si les ensembles représentés par les entités spécialisées sont des ensembles qui n'ont pas d'éléments communs. Dans le cas contraire on parle de non disjonction.

Exemple de disjonction:



Une personne dans un établissement scolaire est soit un étudiant, soit un enseignant, mais une personne ne peut être à la fois étudiant et enseignant.

Contre exemple : non disjonction



Cette spécialisation ne respecte pas la contrainte de disjonction car une occurrence de véhicule peut être à la fois terrestre et à moteur.

C.2 - Les contraintes sur spécialisation

C.2.1 - La contrainte de totalité (T)

Couverture et non Disjonction

Pas d'autre résident à l'hôpital
Le personnel hospitalier réside à l'hôpital.
Un patient réside à l'hôpital
Intersection possible

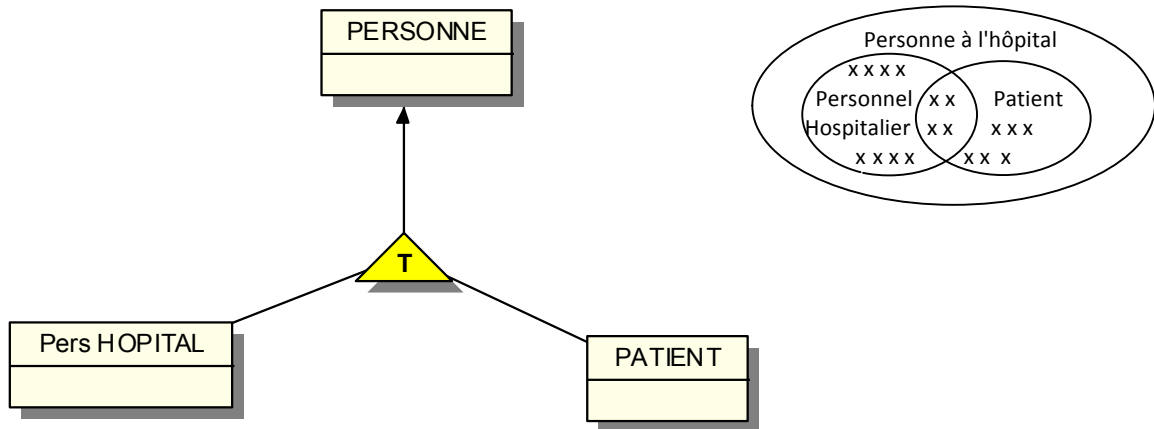
Rappel table de vérité:

A	B	+ inclusif
0	0	0
0	1	1
1	0	1
1	1	1

Interprétation de la table de vérité :

Une personne ni patiente ni faisant partie du personnel n'existe pas

Une personne ne faisant pas partie du personnel, mais patiente existe.
 Une personne membre du personnel et non patiente existe aussi.
 Une personne à la fois membre du personnel et patiente existe aussi



C.2.2 - La contrainte d'exclusion (X)

Non Couverture et Disjonction

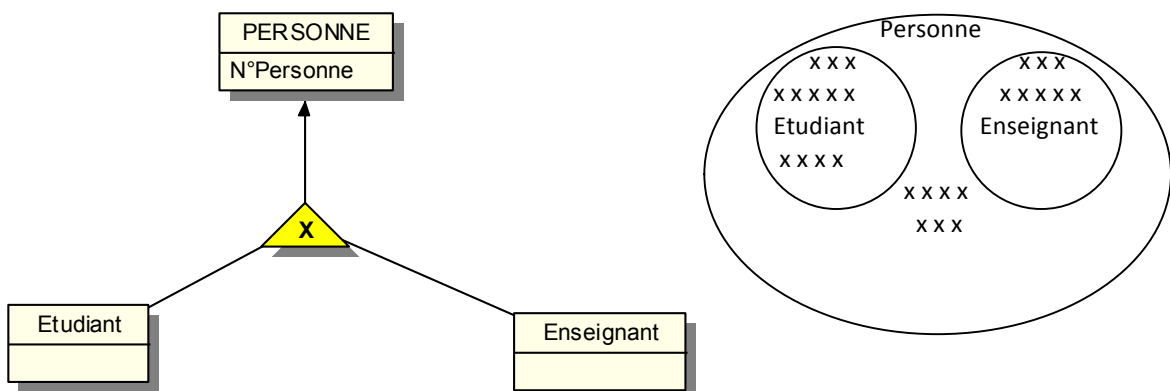
Un étudiant est une personne dans un établissement scolaire
 Un enseignant est une personne dans un établissement scolaire
 Si ce n'est ni un étudiant, ni un enseignant il peut tout de même être une personne dans un établissement scolaire
 Intersection impossible

Table de vérité du non ET (NAND)
 1,1 exclu

A	B	NAND
0	0	1
0	1	1
1	0	1
1	1	imp

Interprétation de la table de vérité :

Une personne sans être étudiant ou enseignante peut exister.
 Une personne non étudiante peut être enseignante donc exister
 Une personne étudiante n'est pas enseignante, elle existe aussi
 Une personne à la fois étudiante et enseignante n'existe pas



Une personne dans un établissement scolaire est soit un étudiant, soit un enseignant, il peut exister d'autres types de personnes.

C.2.3 - La contrainte de partition (XT) ou (+)

Couverture et Disjonction

Un mensuel est un employé. Un vacataire est un employé. L'intersection est impossible. La table de vérité que l'on peut dresser correspond au OU exclusif.

A	B	\oplus
0	0	0
0	1	1
1	0	1
1	1	imp

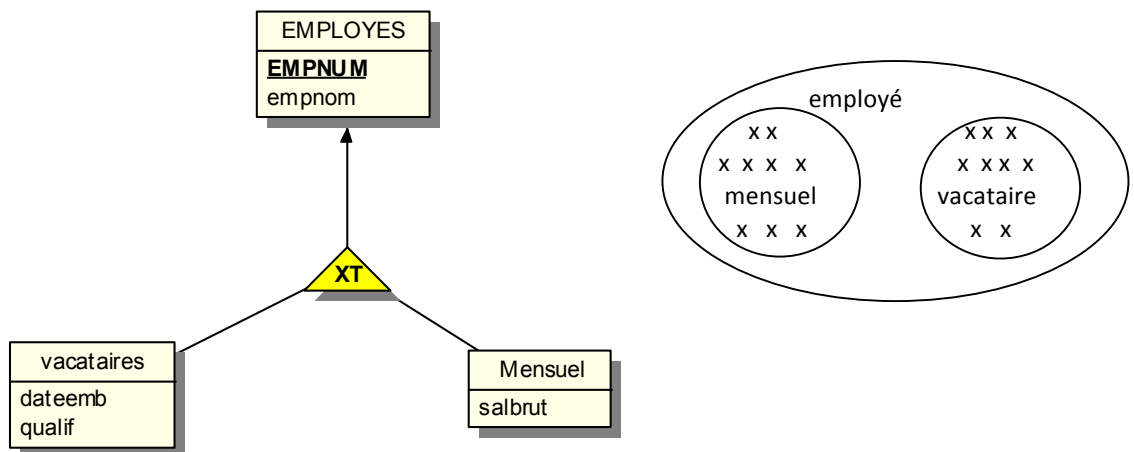
Interprétation de la table de vérité :

Un employé non vacataire et non mensuel n'existe pas

Un employé non vacataire mais mensuel, existe

Un employé vacataire et non mensuel, existe

Un employé à la fois vacataire et mensuel n'existe pas.



C.2.4 - Tableau récapitulatif

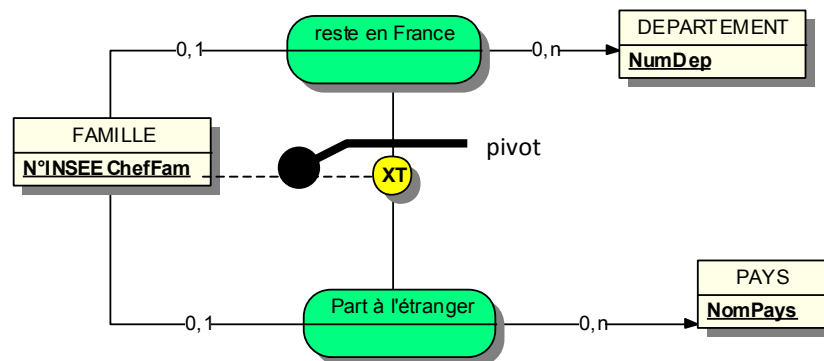
	COUVERTURE	NON COUVERTURE
DISJONCTION		
NON DISJONCTION		

D. Contraintes d'intégrité entre associations

De même que l'on a pu mettre en évidence des contraintes entre entités de façon à préciser notre modèle de données, nous allons maintenant établir les différentes contraintes possibles entre associations à l'aide d'exemples. Nous utiliserons toujours nos contraintes de base couverture et disjonction. La notion de couverture rejoint celle de **T** (Totalité), la notion de disjonction rejoint celle de **X** (eXclusion). La contrainte d'intégrité porte sur 2 ou plusieurs relations.

D.1. contrainte de partition \oplus

Exemple : Une société gère des centres de vacances en France et à l'étranger.
Une famille choisit son lieu de vacances dès son inscription.



Analyse :

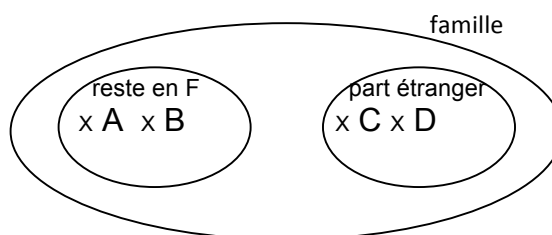
Une famille reste en France dans un département ou part à l'étranger dans un pays. C'est l'un ou l'autre mais pas les deux à la fois (disjonction X). En plus, il n'est pas possible que la famille parte ailleurs que sur terre. S'il n'existe aucune autre solution de voyage, la destination ne peut donc être autre que le département français ou le pays étranger (totalité T).

En résumé :

Le pivot qui relie la contrainte de partition à l'entité famille, signifie, qu'une famille reste en France, ou part à l'étranger, l'un ou l'autre, mais ne peut partir ailleurs, et est forcée de partir.

Représentation :

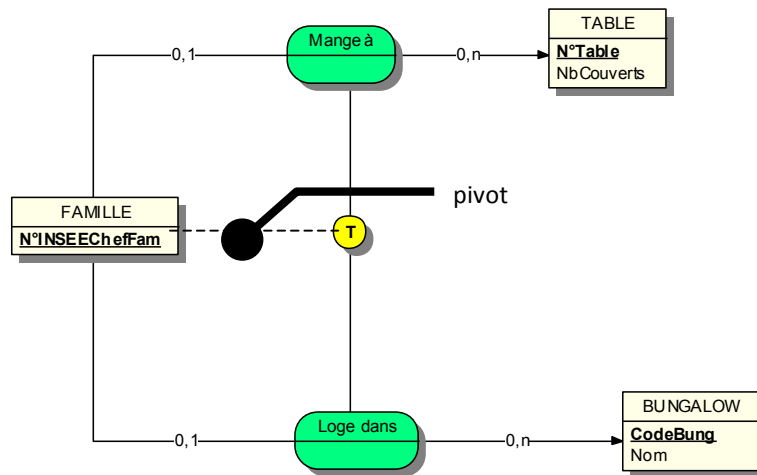
Soit des familles
A, B, C, D



Nous obtenons une couverture et une disjonction, caractéristique de la contrainte de partition XT.

D.2. Contrainte de totalité (T)

Exemple : Chaque centre de vacances propose l'hébergement et la restauration. Dans le cas de l'hébergement la famille choisit son bungalow, dans le cas de la restauration la famille choisit la table et le nombre de couverts. Une famille peut réserver les deux et ne peut en aucun cas s'inscrire au centre sans l'une de ces 2 options.



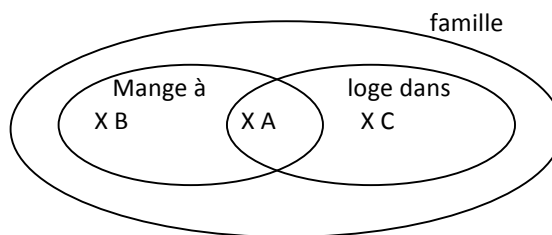
Analyse :

Une famille peut manger à une table et loger dans un bungalow. Une famille peut manger à table ou loger dans un bungalow. Une famille à l'une ou l'autre des possibilités ou bien les deux en même temps. Pas d'exclusion possible donc non disjonction (pas X).

De plus, la famille n'a pas le choix, si elle mange à table, soit elle loge dans un bungalow, soit elle prend les deux options pour s'inscrire. Mais en aucun cas elle ne peut prendre autre chose pour s'inscrire, et en aucun cas elle ne peut ne rien prendre pour s'inscrire, donc totalité (T).

Représentation :

Soit des familles
A, B, C

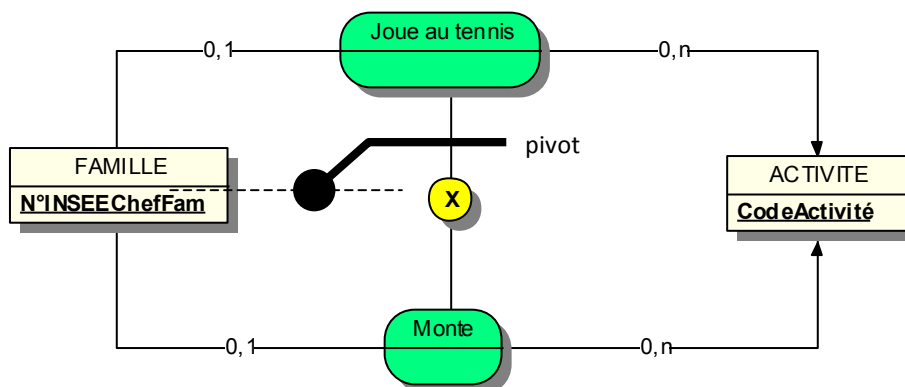


Nous obtenons une couverture et une non disjonction, caractéristique de la contrainte de totalité T.

D.3. contrainte d'exclusion (X)

Exemple 1 : Le centre propose deux activités sportives : Equitation et Tennis.

Une famille peut s'inscrire au plus à l'une des deux. Mais elle peut ne participer à aucune activité.

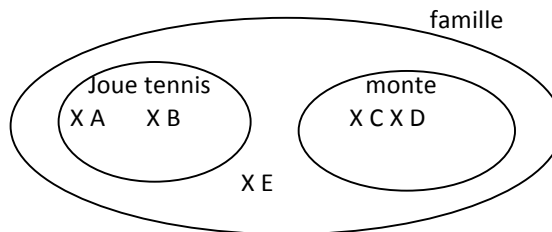


Analyse :

Une famille joue soit au tennis ou monte à cheval. Elle pratique soit l'une ou l'autre des activités, mais pas les deux (disjonction X). Il est possible qu'une famille ne pratique aucune activité, donc pas de couverture (pas de totalité T).

Représentation :

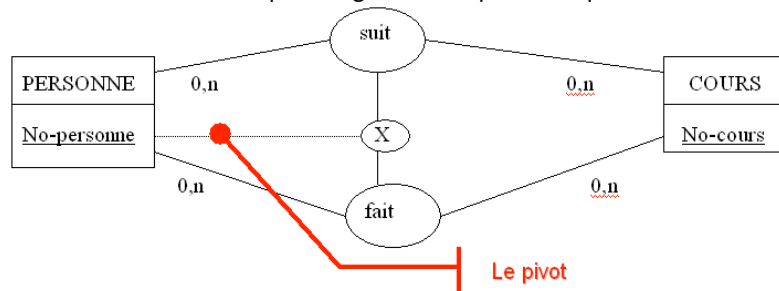
Soit des familles
A, B, C, D, E



Nous obtenons une non couverture et une disjonction, caractéristique de la contrainte d'exclusion X.

Exemple 2 : Une personne fait ou suit des cours

Une personne peut faire des cours en tant qu'enseignant. Une personne peut suivre des cours en tant qu'élève.



Analyse :

Une personne peut faire des cours. Une personne peut suivre des cours. Une personne ne fait pas les deux activités en même temps (exclusion X).

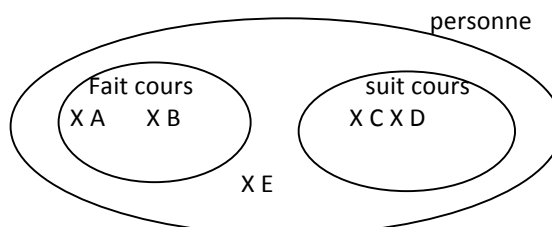
Il est même possible qu'une personne ne fasse aucune des deux activités.

Conclusion :

Le pivot relie la contrainte d'exclusion à l'entité PERSONNE pour signifier : une personne qui fait cours ne suit pas de cours et réciproquement.

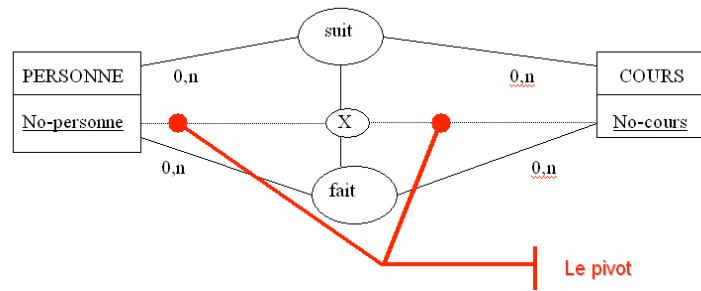
Représentation :

Soit les personnes
A, B, C, D, E



Nous obtenons une non couverture et une disjonction, caractéristique de la contrainte d'exclusion X. On ne fait pas référence au cours.

Exemple 3 : Une personne fait un cours, mais elle ne suit pas le cours qu'elle a réalisé. Dans le cas précédent il n'est pas précisé à quel cours on fait référence.

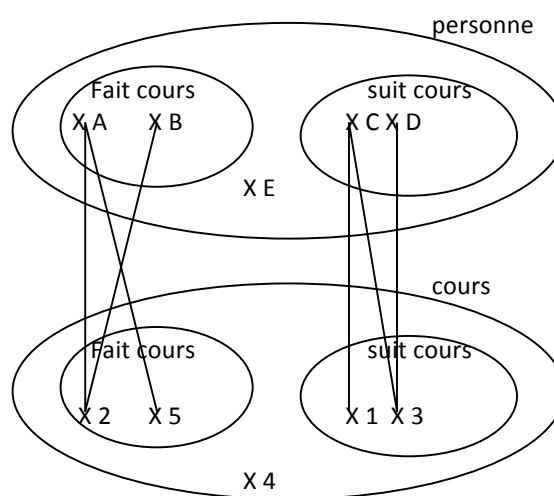


Conclusion :

Le pivot relie la contrainte d'exclusion aux entités PERSONNE et COURS pour signifier : une personne qui fait un cours ne suit pas ce cours et réciproquement.

Représentation :

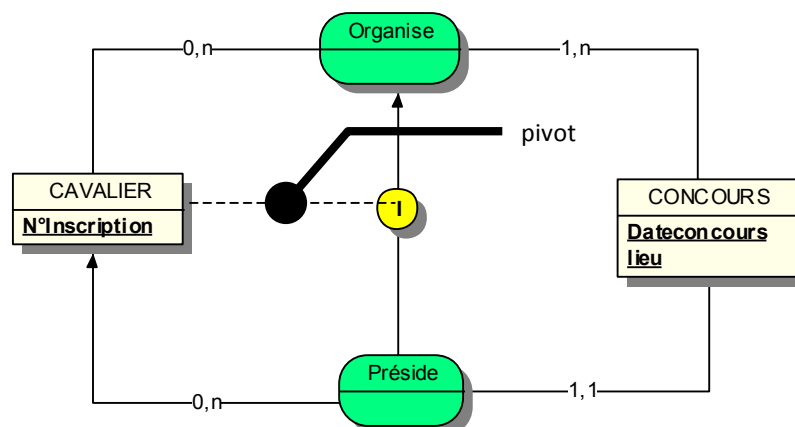
Soit les personnes
A, B, C, D, E



Nous obtenons une non couverture et une disjonction, caractéristique de la contrainte d'exclusion X.

D.4. contrainte d'inclusion (I)

Exemple 1 : la pratique de l'équitation permet de faire des concours qui sont organisés par un comité dont seul un cavalier peut devenir président.

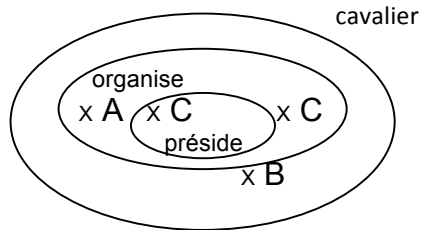


Analyse :

Un cavalier qui préside un concours, organise automatiquement le concours. La réciproque n'est pas forcément vrai.

Représentation :

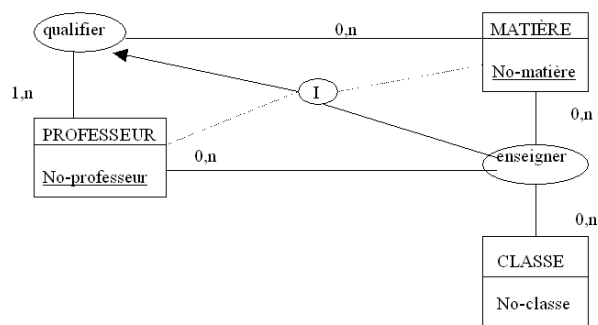
Soit des cavaliers
A, B, C



- B ne fait rien
- A organise
- C préside et organise

Nous obtenons une non couverture et une inclusion, caractéristique de la contrainte d'inclusion I.
Nous ne faisons pas allusion au concours concerné

Exemple 2 :

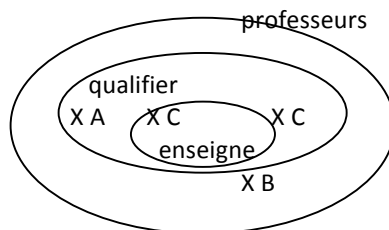


Analyse :

Un professeur qui enseigne une matière dans une classe fait partie des professeurs qualifiés pour cette matière.
Ici, nous faisons allusion à la matière concernée

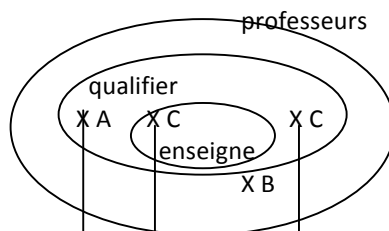
Représentation :

Soit des professeurs
A, B, C



- B n'est pas qualifié et n'enseigne pas une matière
- A est qualifié pour une matière mais ne l'enseigne pas
- C enseigne la matière pour laquelle il est qualifié

Soit des professeurs
A, B, C



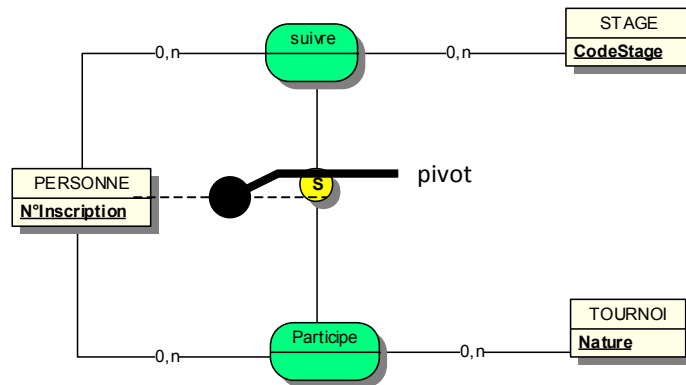
Soit des matières
1, 2, 3



D.5. Contrainte d'égalité (= ou S)

Exemple 1 : toute personne qui suit un stage doit participer au tournoi de clôture du sport qu'il pratique et vice versa.

Les personnes qui participent aux 2 associations sont les mêmes.

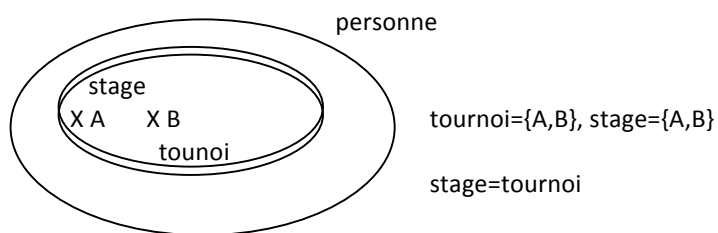


Analyse :

Une personne qui suit un stage, participe au tournoi de clôture de stage. La réciproque est vraie. Une personne qui participe au tournoi de fin de stage, à forcément suivie le stage.

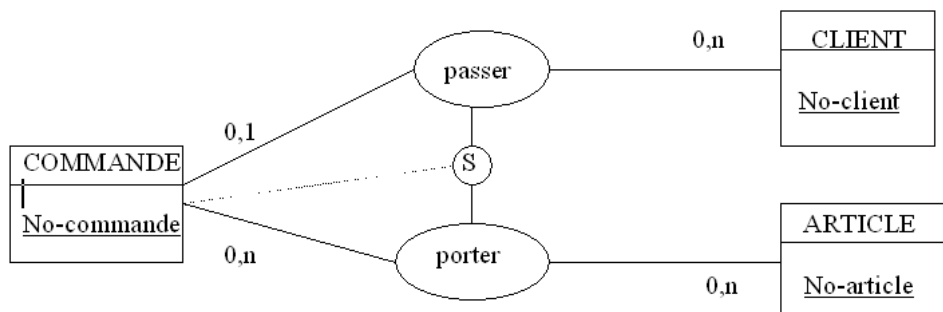
Représentation :

Soit des personnes
A, B



Nous obtenons une couverture et une égalité, caractéristique de la contrainte d'égalité S ou =.

Exemple 2 :

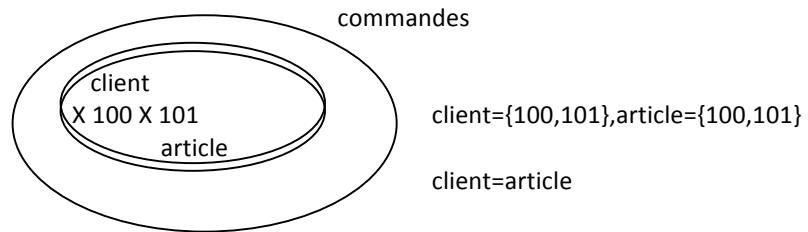


Analyse :

Les commandes qui sont passées par les clients, sont les mêmes qui sont portées par les articles.

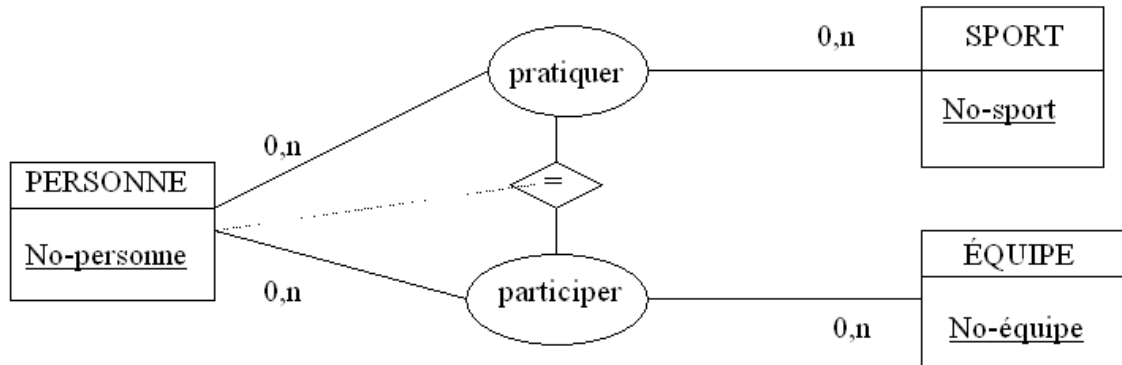
Représentation :

Soit des commandes
100, 101



Nous obtenons une couverture et une égalité, caractéristique de la contrainte d'égalité S ou =.

Exemple 3 :

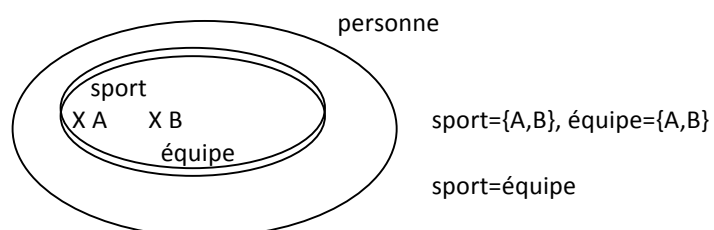


Analyse :

Les personnes qui pratiquent des sports, sont les mêmes qui participent à des équipes.

Représentation :

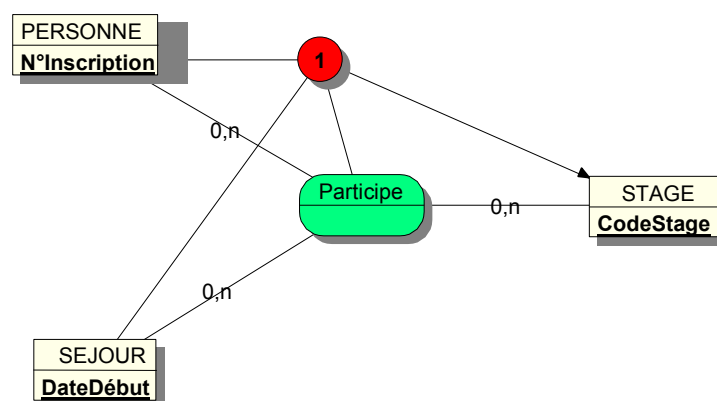
Soit des personnes
A, B



Nous obtenons une couverture et une égalité, caractéristique de la contrainte d'égalité S ou =.

D.6. Contrainte d'unicité (1)

Exemple : Une personne, pendant un séjour donné, ne peut participer qu'à un stage

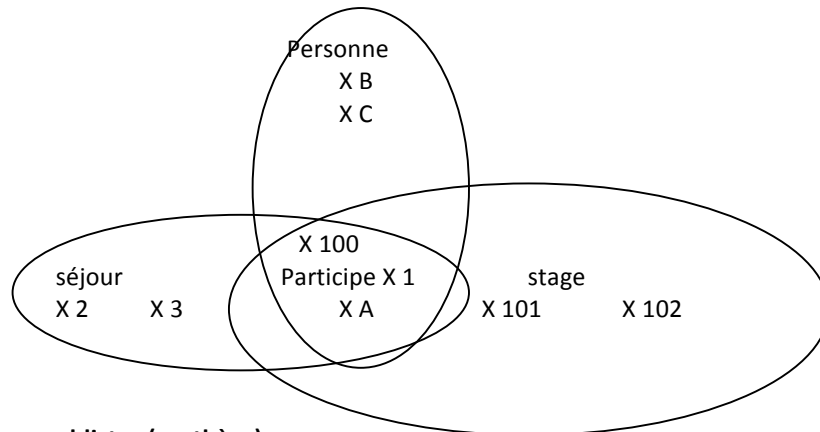


Analyse :

Une personne, pendant un séjour donné, ne peut participer qu'à un stage



Représentation :

Soit les ensembles personne = {A, B, C}, stage = {101, 102, 103}, séjour = {1, 2, 3}



D.7. Les contraintes ensemblistes (synthèse)

Voici un tableau récapitulatif des symboles en vigueur pour représenter les différentes contraintes.

Contraintes à exprimer	Autre notation	Merise/2
Symbole graphique utilisé	Losange 	Cercle 
Totalité	T	Ou inclusif T
Exclusion	X	Exclusivité X
Partition	+	Ou exclusif X T
Inclusion	I	I
Unicité	1	1
Égalité	=	Simultanéité S

E. Prise en compte du temps

E.1. Historisation des Données

L'**historisation** d'une donnée **correspond** à la **conservation** de ses valeurs dans le temps.

Il faut la **distinguer de l'archivage** qui correspond à la **sauvegarde d'occurrences de données**.

Pour chaque donnée à historiser, il est nécessaire de répondre aux questions suivantes :

- qui historise ?
- quelle est la durée d'historisation ?

- quel est le support d'historisation ?

L'**historisation** d'une donnée **intervient** à la fin d'une **phase (MOT)**, c'est-à-dire après sa mise à jour (**CVO**) et le retour du système à un état cohérent.

Plusieurs solutions sont possibles afin d'organiser l'historisation d'une donnée avec toutes les informations convenablement datées :

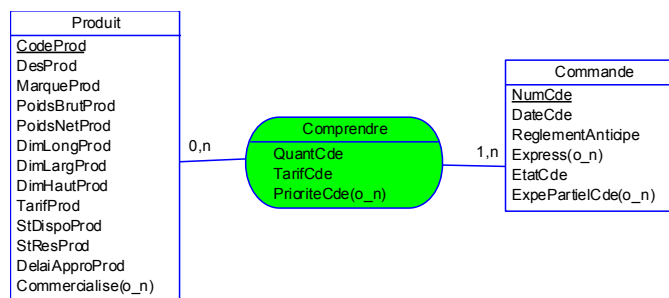
- une donnée (parfois à priori redondante) est ajoutée au modèle,
- une entité Date est créée,
- une propriété mémorisant la donnée est ajoutée au modèle afin d'exprimer l'état des occurrences d'un objet.

Exemple 1 :

La valeur de la propriété TarifCde de l'association Comprendre correspond à la valeur de la propriété TarifProd de l'entité Produit à la date de la commande.

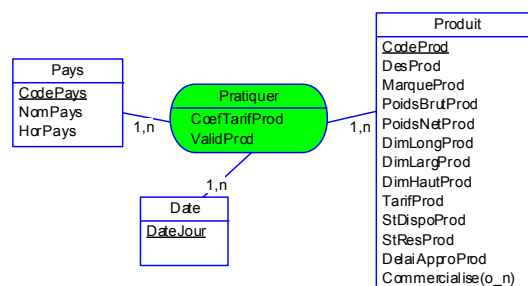
Cette propriété qui paraît a priori redondante permet de conserver le prix du produit (aux fins de facturation, par ex.) à la date de la commande, malgré l'évolution du prix du produit dans le temps.

Exemple 2 :



Le prix unitaire de base de chaque produit est affecté d'un coefficient en fonction du pays de destination pour déterminer le prix pratiqué localement. Ce coefficient peut évoluer dans le temps.

La création d'une pseudo entité "Date" permet d'historiser le "CoefTarifProd" d'un produit appliqué à un pays à une date donnée.



Exemple 3 :

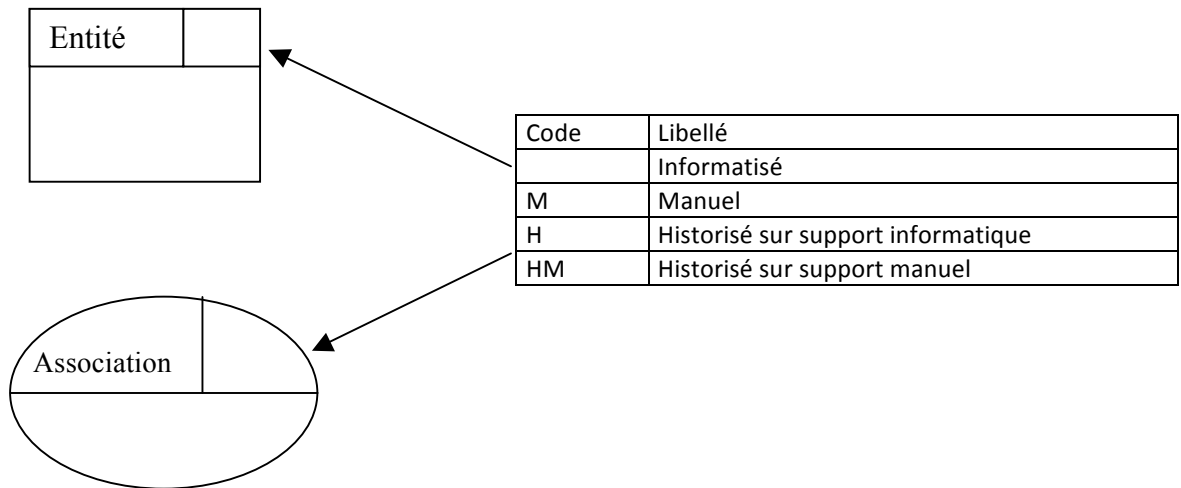


La donnée Commercialise permet d'exprimer le fait que toute référence de produit a une durée de vie : une occurrence de CodeProd naît avec la saisie d'une occurrence de CodeProd dans le système d'information et de la valeur "oui" donnée à Commercialise (o_n). Une valeur de la donnée Commercialise égale à "non" permet de rejeter les commandes comportant cette référence qui n'est plus distribuée (anomalie). Le maintien de la référence dans le système d'information permet en même temps de recalculer une facture antérieure ou une facture en cours. Il est possible de remplacer cette donnée "Commercialise (o_n)" par une donnée "DateFinCommercialise".

E.2. Nature des données

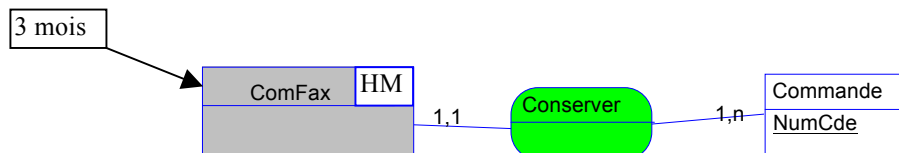
La représentation graphique du schéma des données autorise les descriptions simples de la nature des données. Au-delà des données informatisées, il peut être également pertinent de faire apparaître les données historisées sur support manuel, dans le but d'appréhender précisément le fonctionnement de l'organisation.

Quand cela est possible, la représentation graphique utilise le formalisme suivant :



La plupart des données sont informatisées. Pour ne pas surcharger le modèle, on n'indique aucun code pour les représenter.

Exemple : Les commandes des clients reçues par fax, sont conservées pendant trois mois.



F. Quelques particularités pour le passage au modèle relationnel

F.1. Règles de base

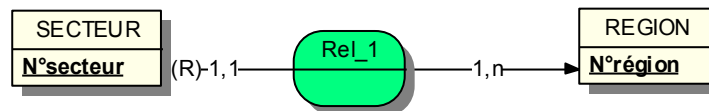
Pour obtenir un modèle relationnel à partir du MEA (Modèle Entité Association), il faut appliquer les règles suivantes :

- les pseudo-entités (attribut spatio-temporel), n'apparaissent plus dans le modèle relationnel (DATE par exemple)
- les entités deviennent des relations dont la clef primaire de la relation est l'identifiant de l'entité.
- les associations deviennent des relations dont la clef primaire est la concaténation des n identifiants des entités concourant à la définition de l'association
- une CIF (lien 1,1-1,n entre 2 entités) se traduit par la migration en clef étrangère dans la relation source de l'identifiant cible du lien d'appartenance

Dans la plupart des **ateliers de génie logiciel (AGL)** (AMC, Win'design) le type de lien 0,1 est traduit par une migration de clef avec une clef étrangère pouvant avoir une valeur nulle.

Les clefs primaires et étrangères de chaque relation composant le modèle logique doivent être mises en évidence (par exemple souligner les clefs primaires et ajouter après les clefs étrangères le symbole #). La légende choisie doit être exposée.

F.2. Entité Forte/entité faible



Dans le cas d'une entité ayant un identifiant relatif, celle-ci devient une relation dont la clef primaire est composée de l'identifiant de l'entité but du lien d'appartenance et de l'identifiant relatif



Soient les tables

REGION (N°Région, ...)

SECTEUR (#N°Region, N°secteur, ...)

F.3 Traduction de la spécialisation/généralisation d'entité

F.3.1. Rappels

Rappel sur la spécialisation d'entité :

La spécialisation d'entité consiste à décomposer une entité dite générique en entités spécialisées pour prendre en compte des caractéristiques particulières à certaines occurrences d'entités :

- ◆ Propriétés spécifiques
- ◆ Liens vers des associations spécifiques.

Rappel sur la généralisation d'entités :

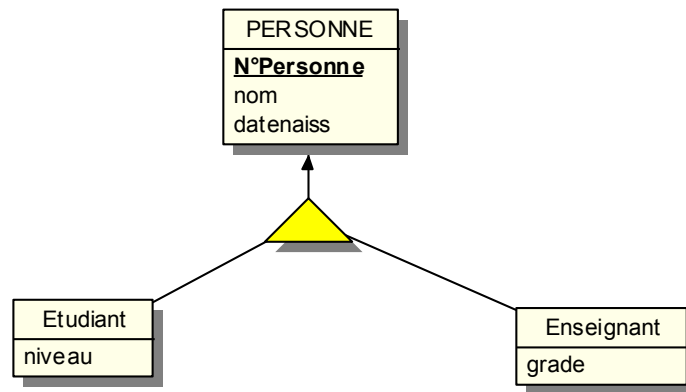
La généralisation permet de regrouper au sein d'une entité générique des entités qui présentent des propriétés communes.

Rappel sur les contraintes entre entités spécialisées :

Deux contraintes de base (la couverture et la disjonction) sont combinées pour exprimer trois types de contraintes sémantiques qui seront représentées graphiquement :

- Exclusion (X)
- Totalité (T)
- Partition (XT)

Dans le cas des entités spécialisées, il existe 4 cas possibles de génération de modèle relationnel que nous allons détailler en prenant l'exemple de spécialisation ci-dessous.



F.3.2. Modèle relationnel

Le passage au modèle relationnel va se faire en cherchant à appliquer des règles qui génèrent le moins de relations et le moins de contraintes possibles.

On va chercher à représenter **sous forme de relations** les **objets** du MEA ayant le **plus de propriétés spécifiques**, et **sous forme de vues** les **objets** du MEA n'ayant **pas ou peu de propriétés spécifiques**.

Une vue est le résultat d'une requête sur une ou plusieurs relations. Une vue ne contient pas d'informations, c'est une photographie permettant de sélectionner les t-uples correspondant à un (ou plusieurs) critères de sélection.

Ces règles vont dépendre du type de contrainte exprimée entre les entités spécialisées.

F.3.2.1- Traduction de chaque entité par une relation (contrainte X entre sous-types)

Chaque entité est traduite par une relation. Toutes les relations ont la même clef primaire qui correspond à l'identifiant de l'entité générique.

Ceci permet de traduire correctement le modèle de données mais ce mode de traduction entraîne la multiplication des tables qui induit des contraintes de mise à jour, notamment lors de l'ajout de n-uplets.

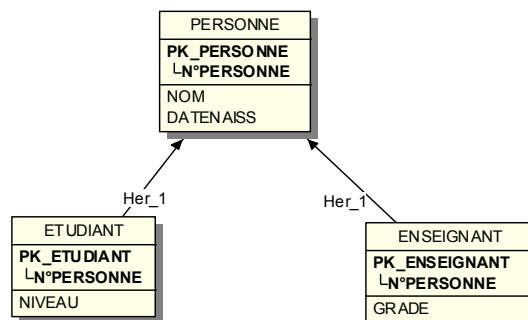
L'emploi de ce type de traduction est **adapté pour un schéma avec une contrainte d'exclusion entre sous-types**.

L'héritage des propriétés du sur-type peut être implémenté de deux façons :

- 1) En réalisant dynamiquement une jointure sur la clef primaire.
- 2) En répétant dans les relations de sous-type les attributs de l'entité générique

1^{ère} solution : En réalisant dynamiquement une jointure sur la clef primaire.

Les tables correspondant au sur-type et aux sous-types sont générées, chacune ayant pour origine l'identifiant du sur type en tant que clé primaire.

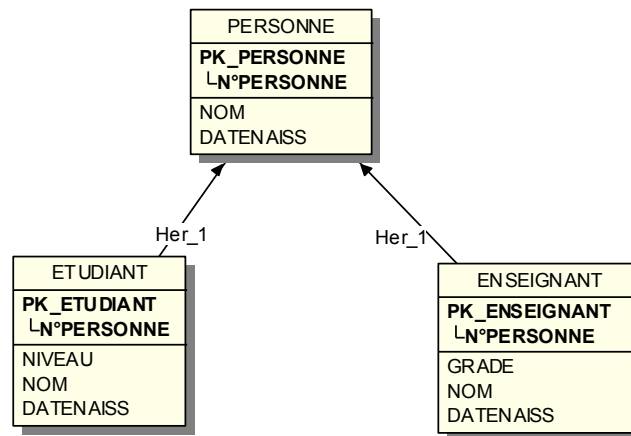


Ainsi pour obtenir le nom d'un étudiant, il faudra réaliser une jointure sur le N°personne entre les tables étudiant et personne.

Remarque : Pour les tables ayant pour origine les sous types, la clé primaire est composée de la clé étrangère ayant pour référence la table ayant pour origine le sur-type.

2^{ème} solution : En répétant dans les relations de sous-type les attributs de l'entité générique

Dans ce cas toutes les propriétés du sur-type sont dupliquées complètement dans chaque sous-type. Cette option aura pour effet de dupliquer tous les attributs de la table mère dans les tables filles.



F.3.2.2- Traduction uniquement des entités spécialisées (contraintes XT ou T entre sous-types)

Seules les entités spécialisées sont traduites sous forme de tables qui héritent des propriétés de l'entité générique.

La table générique sera obtenue par une vue.

Ceci a l'avantage de générer moins de relations et donc d'obtenir moins de contraintes lors des mises à jour.

En revanche ce **type de représentation n'est valable que s'il y a une contrainte de partition ou totalité entre les entités spécialisées.**

L'exclusion permettant que certaines entités génériques ne soient pas représentées par les sous types n'admet pas cette possibilité.

3^{ème} solution : En gardant uniquement les entités spécialisées



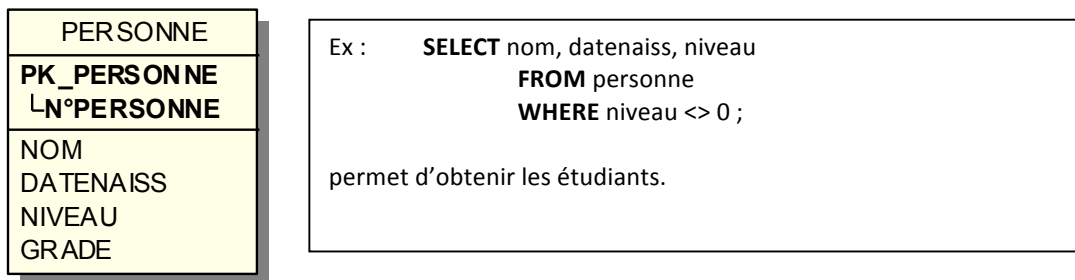
F.3.2.3- Traduction de l'entité générique seulement (pas de contraintes entre sous-types)

L'entité générique devient une relation porteuse des attributs génériques et de l'ensemble des attributs de spécialisation.

Pour certains n-uplets, des attributs auront un sens, pour d'autres pas, et ce en fonction du sous-types d'appartenance : c'est le phénomène d'expropriation de propriété, qui oblige au niveau de la table générique d'autoriser la valeur nulle pour certains attributs.

L'intérêt de ce mode de traduction est la diminution du nombre de tables. Mais il y a un inconvénient majeur qui est la perte sémantique. Seule l'application de vues permet d'obtenir les t-uples correspondant à une entité spécialisée.

4^{ème} solution : En gardant uniquement l'entité générique



F.4. Traduction des contraintes entre entités spécialisées

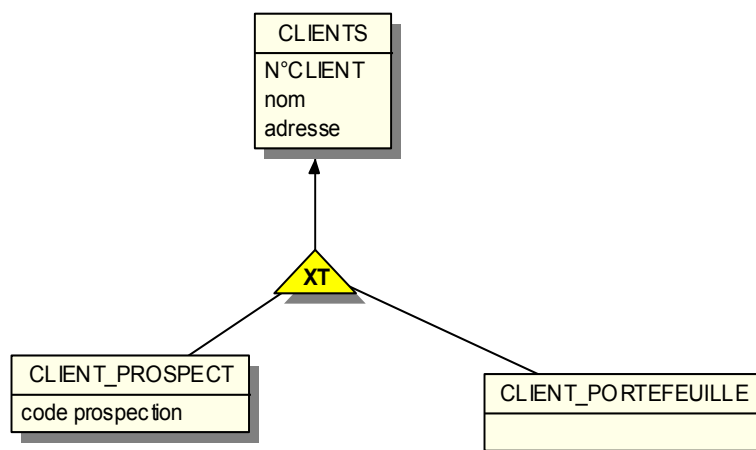
La plupart de ces contraintes se traduisent au niveau du **modèle physique** par l'implantation de triggers.

Un trigger ou déclencheur est un **programme** défini par le programmeur qui s'exécute lors d'un **événement** survenant sur une table d'une base de données. **Un trigger est associé à une seule table.**

L'événement est lié à une demande de mise à jour sur la table (ajout d'un n-uplet, modification ou suppression).

Un trigger est un ensemble de traitement PL/SQL. Un trigger ne peut être appelé directement. Si une table est supprimée, les triggers qui lui sont associés sont automatiquement supprimés.

PL/SQL est un langage procédural (normalisé ANSI/ISO SQL3). Ce langage s'utilise dans des bases de données relationnelles type SQL Server ou ORACLE.



Supposons que le modèle relationnel représentant ce MEA soit déduit des règles : on ne garde que les entités spécialisées (XT).

CLIENT_PROSPECT (code client, nom client, adresse client, code prospection)

CLIENT_PORTEFEUILLE (code client, nom client, adresse client)

Exemple de triggers attaché à l'opération d'ajout d'un t-uple dans la table CLIENT_PROSPECT (SGBD Oracle 7) permettant de vérifier la disjonction entre sous-types :

```
CREATE TRIGGER client_prospect_INSERT
BEFORE INSERT ON client_prospect
FOR EACH ROW
DECLARE
    check_val integer;
    erreur exception;
BEGIN
    -- vérification disjonction
    -- compte toutes les lignes de la table client_portefeuille qui comportent le numéro client inséré
    SELECT COUNT(*) INTO check_val FROM client_portefeuille
    WHERE
        client_portefeuille.code_client = :new.code_client;
    -- new référence la nouvelle valeur d'une colonne
    -- old référence l'ancienne valeur d'une colonne
    -- si plus de 0 lignes, alors le numéro existe déjà donc erreur
    IF check_val != 0
    THEN
        raise erreur
    ELSE
        RETURN;
    END IF;
exception
when erreur then
    errno:=-20002;
    errmsg:='enregistrement existant';
    raise_application_error(errno,errmsg) ;
END;
/
```

Ce trigger permet de vérifier que l'insertion d'un nouveau CLIENT_PROSPECT est cohérente.

En effet la représentation du modèle entité association indique une partition entre les sous-type CLIENT_PROSPECT et CLIENT_PORTEFEUILLE un client ne peut donc appartenir aux deux sous-types.

On va regarder qu'il n'existe pas de t-uple dans la relation CLIENT_PORTEFEUILLE portant ce numéro avant d'insérer le t-uple dans la relation CLIENT_PROSPECT.

S'il existe un t-uple, une erreur sera générée et l'ajout ne sera pas possible.

6. DPD et SQL, une (grande) parenthèse s'impose !

Le **langage SQL (Structured Query Language)** est en passe de devenir le standard en matière d'interface relationnelle, ceci probablement à cause de deux raisons :

- issu de **SEQUEL** (interface de System-R), **SQL** a été développé chez **IBM** à San José !
- **basé** sur des **mots clefs anglais explicites**, il est relativement **simple** et facile à apprendre pour des utilisateurs non-informaticiens. Il illustre bien la tendance des **langages formels** à s'orienter vers un certain "**langage naturel**" (**L4G**).

Il existe de **nombreux dialectes SQL**, la plupart des versions commercialisées sont appauvries par rapport à **SEQUEL (Structured English as a QUery Langage)**.

SQL est un langage relationnel qui a une triple dimension :

- **interrogation** et **modification des occurrences** d'une base de données relationnelle (LMD : langage de manipulation ; INSERT, UPDATE, LID : langage d'interrogation ; SELECT)
- **définition** et **modification du schéma d'une base** de données relationnelle (LDD : langage de définition de données; CREATE)
- contrôle de **sécurité** et **d'intégrité de la base** (LCD : langage de contrôle des données; GRANT)

Pour faire le lien avec notre cours MERISE seule la définition de données (LDD) nous intéresse ici. Le reste constitue une littérature sur laquelle nous reviendrons lors de l'approche complète et concrète d'un « vrai » SGBDR.

SQL est aussi bien un **langage interactif** qu'un **langage intégré** ("embedded") dans un langage de programmation (C ou Cobol, ...) pour le développement d'applications.

Cependant SQL ne constitue pas (à l'exception du standard de référence assez peu suivi) un langage unique, mais un ensemble de dialectes.

6.1. SQL un Langage de Définition des Données (LDD)

À partir d'un modèle relationnel en troisième forme normale (3FN) et après une éventuelle optimisation de ce schéma, vous pouvez créer votre base de données à l'aide de votre SGBD Relationnel incorporant généralement le langage SQL. Cette étape s'inscrit dans la DPD.

Vous pouvez créer autant de base de données que vous voulez, la seule limite étant l'espace disque dont vous disposez sur votre ordinateur.

1. Définir votre Base de Données
2. Créer vos Tables et Index
3. Définir les autorisations d'accès à la BdD
4. Saisir les Informations de la BdD
5. Mettre la BdD à la disposition des utilisateurs
6. Vérifier la protection des informations

Nous allons étudier les différents ordres SQL associés aux deux premières étapes en indiquant les instructions spécifiques au SGBD Relationnel ORACLE V7. La version actuelle d'ORACLE est la version 9i (dédiée internet et orientée d'objet). Les versions ORACLE sont orientées objets à partir de la version 8.

6.3. Créer et ouvrir une base de données

Des ordres SQL spécifiques permettent à l'administrateur de la base de données de monter, d'ouvrir et de fermer des bases de données. Ces fonctions sont disponibles à partir de Sqldba (Administrateur base de données) uniquement.

La gestion des différentes bases de données avec le SGBD Oracle est effectuée avec des verbes SQL, mais est lié d'une part à l'environnement de l'utilisateur sur son système d'exploitation et d'autre part aux droits d'accès de l'utilisateur Oracle.

La création, l'ouverture et la fermeture de la base sont assurés par l'administrateur de la base de données, qui pour automatiser ces procédures peut utiliser des programmes dépendant directement du système d'exploitation de la machine.

Avec la version 7 d'Oracle, l'administrateur doit se connecter avec Sqlldr :

Connect Internal

Pour l'ouverture et la fermeture d'une base de données il utilise les commandes suivantes.

- l'ordre d'ouverture est : **Startup**,
- l'ordre de fermeture est : **Shutdown**.

La première création se fera par l'ordre SQL :

```
CREATE DATABASE nom de base  
...  
LOGFILE 'nom fichier1' SIZE integer [K | M]  
          'nom fichier2' SIZE integer [K | M]  
DATAFILE 'nom fichier3' SIZE integer [K | M]  
...
```

Cet ordre nécessite les privilèges de DBA et supprime le contenu des fichiers spécifiés. Le nom de base est limité à 8 caractères. Le nom de la base par défaut est spécifié dans le fichier Init.ora par la variable DB_NAME.

A l'installation du produit Oracle, une base par défaut est créée automatiquement.

6.4. Créer une Table (la DPD proprement dite)

6.4.1. La syntaxe SQL de base

Avant de créer une table il faut définir les différentes colonnes de la table nécessaires lors de la création. Pour créer une table utilisez l'ordre suivant :

```
CREATE TABLE nom de la table  
(nomcolonne1 type [DEFAULT expr ] [not null],  
 nomcolonne2 type [DEFAULT expr ] [not null],  
  ...  
)  
[PCTFREE entier] [PCTUSED entier]  
[INITTRANS entier] [MAXTRANS entier]  
[TABLESPACE nomtablespace]  
[STORAGE clause de stockage]  
| [CLUSTER nomcluster (colonne [,colonne] ...)]  
[AS requête ]  
;
```

nom colonne : nom de chaque colonne (attribut) de la table à ne pas dupliquer dans une table ;

type : peut appartenir à la liste (non exhaustive) ci-dessous

Numérique : **NUMBER** ou (Decimal, Double Precision, Float, Integer, Real)

syntaxe : Number[longueur,[décimale]]

exemple : salaire number(8,2) : 8 chiffres dont 2 après la virgule.

Par défaut 38 chiffres.

Caractères : **CHAR** et (**VARCHAR** ou **VARCHAR2**)

CHAR : chaîne de caractères de longueur fixe avec un maximum de 255 caractères

Syntaxe : CHAR(n) Exemple : nom CHAR(20)

Par défaut la longueur est égale à 1.

VARCHAR2 : chaîne de caractères de longueur variable avec un maximum de 2000 caractères.

Syntaxe : VARCHAR2(n) Exemple : note VARCHAR2(1000)

VARCHAR : équivalent à VARCHAR2 avec Oracle 7. Mais différent dans les prochaines versions. A ne pas utiliser.

Chaîne longue : LONG et LONG RAW

Syntaxe : LONG Exemple : explication LONG

Chaîne de caractères de 64ko maximum (Oracle 6)

Chaîne de caractères jusqu'à 2 giga octets (Oracle 7)

Une colonne Long par table.

Inutilisable dans une expression, un prédicat ou un ordre ORDER BY.

Date : DATE

Syntaxe : DATE

Exemple : datecommande date

Pas de longueur spécifiée, format interne fixe avec stockage du siècle, de l'année, du mois, du jour, de l'heure, des minutes et des secondes.

Le format par défaut est : 'JJ-MMM-AA' ('01-JAN-93')

Une date occupe 7 octets dans la base.

Binaire : RAW

Syntaxe : RAW(longueur) Exemple : octet(8)

Chaîne d'octet de longueur variable manipulée sous forme hexadécimale.

Stockée dans la base comme le type Char.

Identifiant Interne d'une ligne : ROWID

Identifiant de longueur fixe sur 6 octets qui indique le numéro de ligne, le numéro de bloc et le numéro de fichier physique.

Autres types possibles : Decimal, Float, Integer, Real, SmallInt

Exemple : Ordre SQL de création de la table Elève

```
create table ELEVE
(
  CODE_ELEVE          CHAR(5),
  NOM_ELEVE           VARCHAR2(25),
  PRENOM_ELEVE        VARCHAR2(25),
  DATE_NAISS          DATE,
  AGE                  NUMBER(3),
  CLASSE               CHAR(3)
);
```

6.4.2. La syntaxe SQL des contraintes d'intégrité

La prise en compte des contraintes d'intégrité structurelles et applicatives (check) peut être effectuée dans l'ordre de création d'une table relationnelle avec Oracle V7.

En effet, des contraintes exprimées sous forme **déclarative** (non procédurale) permettent de gérer l'intégrité des données de manière sûre et transparente sans aucune programmation.

L'intégrité des données avec Oracle peut également être gérée en écrivant des triggers. Dans ce cas on parle d'intégrité **programmée**.

Les contraintes déclaratives sont utilisées pour :

- vérifier une contrainte d'intégrité applicative ou l'intégrité de domaine : (**CHECK**) ;
- affecter une valeur par défaut à une colonne : (**DEFAULT**)
- vérifier qu'une colonne ou un groupe de colonne possède une valeur (**NOT NULL**)
- vérifier l'unicité de la valeur d'une colonne ou d'un groupe de colonnes (**UNIQUE**) (null autorisé)
- identifier une colonne ou un groupe de colonnes comme clé primaire (**PRIMARY KEY**) ;
- vérifier l'intégrité référentielle par la clause :
- (**FOREIGN KEY REFERENCES ...[ON DELETE CASCADE]**) ;

Elles peuvent être définies directement sur une colonne ou sur la table. Les contraintes de colonnes et les contraintes de tables se différencient par la syntaxe d'écriture et par leur utilisation.

Syntaxe de définition d'une contrainte de colonne :

```
[Constraint nomdelacontrainte]
{ [not null],
  [unique | primary key],
  [references [schema.]table (colonne)
    [on delete cascade]
  [check (condition) ]
}
{using index [pctfree entier] [initrans entier]
  [maxtrans entier]
  [tablespace nomdutablespace]
  [storage clausestorage]
[exceptions into [schema.]table]
disable }
```

Les contraintes de tables concernent un ensemble de colonnes de la table. La première forme ne permet pas de définir une clé unique, primaire ou étrangère composée.

Syntaxe de définition d'une contrainte de table :

```
[Constraint nomdelacontrainte]
{ [unique | primary key] (colonne [,colonne] ...)
  [foreign key (colonne [,colonne] ...)
  references [schema.]table (colonne [,colonne] ...)
    [on delete cascade]
  ]
  [check (condition) ]
}
{using index [pctfree entier] [initrans entier]
  [maxtrans entier]
  [tablespace nomdutablespace]
  [storage clausestorage]
[exceptions into [schema.]table]
disable }
```

Dans les deux cas le nommage de la contrainte est optionnel et sert à attribuer à la contrainte à définir, qui sera sauvegardée dans le dictionnaire de données.

Si le nom est omis, Oracle génère automatiquement un identifiant sous la forme SYS_Cnnnnn où nnnnn représente un numéro unique dans la base de données.

6.4.2.1. La contrainte d'intégrité (CHECK)

La contrainte **check** se caractérise par la mise en place d'un test. Pour satisfaire la contrainte de domaine, toutes les lignes de la table doivent satisfaire le test ou l'ignorer (comparaison avec une valeur inconnue "null").

Règles de la contrainte check :

- Elle peut référencer une ou plusieurs colonnes,
- Elle peut comparer des colonnes entre elles,
- Elle peut comparer des colonnes et des constantes,
- Ne peut pas contenir de sous-requête,
- Ne peut pas contenir les variables SYSDATE et USER.

Exemple 1 : Syntaxe avec contraintes de colonnes

```
Create Table Theme
( Num_theme    number(4),
  Nom_theme    char(20)
    constraint cid_nomtheme
    check (Nom_Theme = upper(Nom_Theme)),
  Date_du_theme date
    constraint cid_datetheme
    check (Date_du_theme > '01-JAN-95')
);
```

Exemple 2 : Syntaxe avec contraintes de tables

```
Create Table Theme
( Num_theme    number(4),
  Nom_theme    char(20),
  Date_du_theme date ,
    constraint cid_nomtheme
    check (Nom_Theme = upper(Nom_theme)),
    constraint cid_datetheme
    check (Date_du_theme > '01-JAN-95')
);
```

Exemple 3 : Gestion d'une liste de valeurs

```
Create Table Type_Appt
( Code_Appt    char(2),
  Lib_Appt     char(20),
    constraint cid_codeappt
    check (Code_Appt in ('F1','F2','F3', 'F4','F5','T1') )
);
```

Exemple 4 : Contrainte de calcul :

```
Create Table Salarie
( Num_salarie    number(4),
  Nom_salarie    char(20),
  Salaire        number (7,2),
  Prime          number (7,2),
  check (Salaire + Prime < 50000),
);
```

Remarques :

Lorsque le nom de la contrainte n'est pas spécifié, Oracle génère automatiquement un nom de contrainte : SYS_Cnnn.

6.4.2.2. La contrainte de valeur nulle ([NOT] NULL)

La contrainte NOT NULL permet d'indiquer qu'une colonne ne peut pas contenir de valeur nulle. Pour satisfaire cette contrainte chaque ligne de la table doit posséder une valeur pour cette colonne.

Par défaut une colonne peut contenir des valeurs nulles.

La définition des colonnes non nulles doit se faire par une contrainte de colonne dans l'instruction Create Table ou Alter Table.

Exemple 1 : contraintes de valeurs nulles nommées

```
create table ELEVE
(
  CODE_ELEVE  CHAR(5)    constraint nn_code_eleve not null,
  NOM_ELEVE   VARCHAR2(25) constraint nn_nom_eleve  not null,
  PRENOM_ELEVE VARCHAR2(25) constraint nn_pre_eleve not null,
  DATE_NAISS  DATE,
  AGE         NUMBER(3),
  CLASSE      CHAR(3)
);
```

6.4.2.3. La contrainte d'unicité (UNIQUE)

La contrainte d'unicité désigne une colonne ou un groupe de colonnes en tant que clé unique (pas de doublon autorisé).

Pour satisfaire cette contrainte deux lignes de la table ne peuvent pas avoir la même valeur pour la clé unique.

Un **index est automatiquement créé** avec comme nom celui de la contrainte. Il est possible d'orienter la création de l'index avec la clause USING INDEX (tablespace différent pour la table et l'index).

Par contre la clé unique composée d'une seule colonne peut posséder des valeurs nulles.

Une clé unique ne peut pas être du type LONG ou LONG RAW.

Pour définir les clés uniques vous pouvez utiliser les contraintes de colonnes et de tables.

Exemple 1 : clé unique définie par une contrainte de colonne

```
Create Table Theme
( Num_theme    number(4) not null,
  Nom_theme    char(20) constraint unq_nom_theme UNIQUE,
  Date_du_theme date ,
  constraint cid_nomtheme
    check (Nom_Theme = upper(Nom_theme),
  constraint cid_datetheme
    check (Date_du_theme > '01-JAN-95')
);
```

Exemple 2 : clé unique composée définie par une contrainte de table

```
create table ELEVE
(
  CODE_ELEVE CHAR(5) constraint nn_code_eleve not null,
  NOM_ELEVE VARCHAR2(25) constraint nn_nom_eleve not null,
  PRENOM_ELEVE VARCHAR2(25) constraint nn_pre_eleve not null,
  DATE_NAISS DATE,
  AGE NUMBER(3),
  CLASSE CHAR(3),
  constraint unq_nom_prenom UNIQUE (NOM_ELEVE,PRENOM_ELEVE);
```

6.4.2.4. La contrainte de clé primaire (PRIMARY KEY)

La contrainte PRIMARY KEY permet de désigner une colonne ou un groupe de colonnes comme clé primaire de la table. Pour satisfaire la contrainte de clé primaire les deux conditions ci-dessous doivent être vérifiées :

- Aucune valeur de clé primaire ne peut apparaître dans plus d'une ligne de la table,
- Aucune colonne qui fait partie de la clé primaire ne contient de valeur nulle.

Règles de la contrainte Primary Key :

- Une table ne peut posséder qu'une seule clé primaire.,
- Une clé primaire ne peut pas être du type LONG ou LONG RAW.,
- Elle peut être définie par une contrainte de colonne ou de table,
- **Création automatique d'un index primaire** avec la possibilité d'orienter la création de l'index avec la clause USING INDEX.

Exemple 1 : clé primaire définie par une contrainte de table

```
Create Table Theme
( Num_theme number(4) not null,
  Nom_theme char(20) constraint unq_nom_theme UNIQUE,
  Date_du_theme date ,
  constraint cid_nomtheme
    check (Nom_Theme = upper(Nom_theme),
  constraint cid_datetheme
    check (Date_du_theme > '01-JAN-95'),
  constraint pk_theme
    PRIMARY KEY (Num_Theme)
    using index tablespace iappli01
);
```

Exemple 2 : clé primaire composée

```
Create Table Ligne_commande
( Numero_cde number not null,
  Numero_lig number not null,
  Quantite number not null,
  PU number not null,
  constraint pk_ligne_cde
    PRIMARY KEY (Numero_cde,Numero_lig)
);
```

6.4.2.5. La contrainte d'intégrité référentielle (FOREIGN KEY)

A. Principe général

Le type d'opération de suppression, modification à appliquer à une valeur de clé étrangère doit être déclaré dans la définition de la table associée.

3 stratégies sont possibles. Elles seront illustrées par le lien entre la table Pilote et la Table Vol.

1. **propagation** (delete en cascade):

Suppression d'une ligne dans la table PILOTE avec suppression de toutes les lignes dans la table VOL concernant le pilote à supprimer.

2. **interdiction** (restrict) :

Suppression interdite d'une ligne dans la table PILOTE s'il existent des lignes dans la table VOL pour ce pilote (la valeur de la clé étrangère est égale à la valeur de clé primaire).

3. **mise à nulle** (set null) :

Dans la table VOL, les valeurs des clés étrangères sont mises à "nulles". La suppression dans la table Pilote est autorisée.

Cette stratégie impose que les clés étrangères ne soient pas déclarées en Not Null.

B. Stratégies offertes par Oracle

Le SGBDR Oracle permet de définir un lien entre une clé primaire (définie par PRIMARY KEY) et une clé étrangère (définie par FOREIGN KEY) avec les instructions :

- Create Table,
- Alter Table.

La définition de ce lien permet de distinguer la table "enfant" (qui contient la clé étrangère) et la table "parent" (qui contient la clé primaire).

Règles de la contrainte FOREIGN KEY :

- La clé primaire ou unique doit être définie avant,
- Les valeurs peuvent être :
 - soit une valeur NULL,
 - soit une valeur existante dans la table référencée.
- La **cohérence** entre les clés primaires et les clés étrangères est **garantie automatiquement**,
- La clé étrangère peut référencer une colonne dans la même table,
- Les deux tables doivent être dans la même base de données, pas de référencement à une table distante.

Restrictions de la contrainte FOREIGN KEY :

Oracle applique la stratégie de restriction par défaut.

- Un enregistrement ne peut pas être supprimé s'il est référencé par une clé étrangère.
- La colonne référencée par la clé étrangère ne peut pas être mise à jour.

L'option ON DELETE CASCADE :

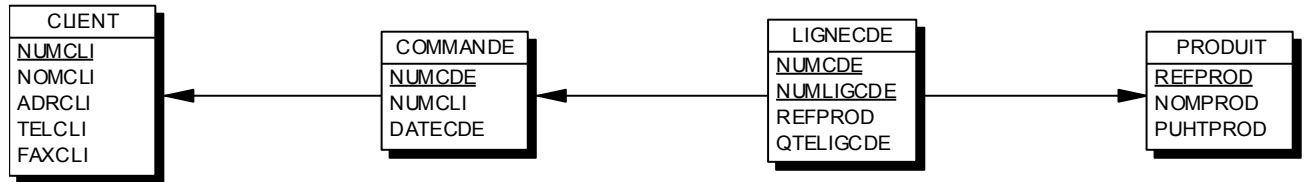
Cette option permet d'appliquer la stratégie de propagation.

- Si une ligne "maître" est supprimée, les lignes référencées par les clés étrangères sont supprimées automatiquement.

- Le message "n rows deleted" ne prend en compte que les lignes de la table "maître".

2.3.7.3. Exemple de gestion commerciale

Le modèle physique des données est le suivant :



Les flèches correspondent aux contraintes d'intégrité référentielle qu'il faut déclarer.

Exemple : Création des tables clients et commandes.

```

create table CLIENT
( NUMCLI      NUMBER(4)      not null,
  NOMCLI      VARCHAR2(35)   not null,
  ADRCLI      VARCHAR2(100)   ,
  TELCLI      VARCHAR2(15)    ,
  FAXCLI      VARCHAR2(15)    ,
  constraint pk_client primary key (NUMCLI)
);

create table COMMANDE
( NUMCDE      NUMBER(8)      not null,
  NUMCLI      NUMBER(4)      not null,
  DATECDE     DATE          not null,
  constraint pk_commande primary key (NUMCDE),
  constraint fk1_commande foreign key (NUMCLI)
    references CLIENT (NUMCLI)
);
  
```

C. Gestion des scripts de création des tables

Lorsque l'administrateur de la base utilise les contraintes d'intégrité référentielles il doit ordonnancer les ordres de création des tables en commençant par les tables **statiques** puis terminer avec les tables **dynamiques**.

Une **table est statique** lorsque aucune colonne n'est clé primaire dans une autre **table**. Elle ne contient pas de clé étrangère.

Dans certains cas, il est **impossible d'ordonnancer** les ordres de création des tables (contrainte référentielle mutuelle).

Exemple :

Un membre appartient à une équipe.
Une équipe possède un membre chef d'équipe.

L'administrateur a deux solutions à sa disposition :

1. Supprimer une clé étrangère pour obtenir une **table** statique et créer une **table** supplémentaire correspondant au lien de la clé étrangère.

2. Créer toutes les tables relationnelles sans utiliser les contraintes référentielles dans l'ordre de création de table.

Après la création de toutes les tables, modifier les tables en rajoutant les contraintes référentielles par l'ordre SQL :

```
Alter Table nom de table ADD  
( CONSTRAINT fk_table FOREIGN KEY (cleetra)  
      REFERENCES tablep (cleprim),  
  ...  
  CONSTRAINT nomcontrainte      ,  
);
```

A l'inverse, lorsque l'utilisateur désire supprimer une table (statique) qui est référencée par une autre table (dynamique), l'**ordre d'enchaînement** des instructions de **suppression des tables** devra être **inversé par rapport à l'ordre des instructions de création**.

On supprime d'abord les **tables** dynamiques,
puis les **tables** statiques.

On peut également supprimer les contraintes par l'instruction :

```
Alter Table nom de table DROP  
UNIQUE (colonne, )  
PRIMARY KEY  
CONSTRAINT nomcontrainte  
[CASCADE]  
;
```

Exemple : Script de création de la gestion commerciale.

Type	Script SQL
Statique	<pre> create table CLIENT (NUMCLI NUMBER(4) not null, NOMCLI VARCHAR2(35) not null, ADRCLI VARCHAR2(100) , TELCLI VARCHAR2(15) , FAXCLI VARCHAR2(15) , constraint pk_client primary key (NUMCLI)); create table PRODUIT (REFPROD CHAR(8) not null, NOMPROD VARCHAR2(40) not null, PUHTPROD NUMBER(8,2) not null, constraint pk_produit primary key (REFPROD)); </pre>
Dynamique	<pre> create table COMMANDE (NUMCDE NUMBER(8) not null, NUMCLI NUMBER(4) not null, DATECDE DATE not null, constraint pk_commande primary key (NUMCDE)); create table LIGNECDE (NUMCDE NUMBER(8) not null, NUMLIGCDE NUMBER(2) not null, REFPROD CHAR(8) not null, QTELIGCDE NUMBER(3) not null, constraint pk_lignecde primary key (NUMCDE, NUMLIGCDE)); alter table COMMANDE add constraint fk1_commande foreign key (NUMCLI) references CLIENT (NUMCLI); alter table LIGNECDE add constraint fk1_lignecde foreign key (NUMCDE) references COMMANDE (NUMCDE); alter table LIGNECDE add constraint fk2_lignecde foreign key (REFPROD) references PRODUIT (REFPROD); </pre>

6.4.2.6. La contrainte DEFAULT

La valeur par défaut sur la colonne sera prise si elle n'est pas spécifiée au moment de la création d'une ligne dans une table et uniquement au moment de la validation.

Cette valeur par défaut peut :

- être une constante appropriée au type de données,
- être une variable du type USER, SYSDATE, etc,...
- être une séquence.

La valeur par défaut ne peut pas :

- inclure une référence à d'autres colonnes,
- utiliser les fonctions PL/Sql.

Exemple :

```
Create Table Theme
( Num_theme    number(4) not null,
  Nom_theme    char(20) constraint unq_nom_theme UNIQUE,
  Date_du_theme date DEFAULT TRUNC(SYSDATE),
  constraint cid_nomtheme
      check (Nom_Theme = upper(Nom_theme),
  constraint cid_datetheme
      check (Date_du_theme > '01-JAN-95'),
  constraint pk_theme PRIMARY KEY (Num_Theme)
);
```

6.4.3. Activation et désactivation des contraintes d'intégrité

On peut activer ou inhiber une contrainte d'intégrité.

6.4.3.1. Activées les contraintes seront contrôlées

- Maintien des verrous sur les tables tant que les contrôles ne sont pas terminés (pas de modification des lignes),
- Contrôle de transgression des lignes,
- Rapports des erreurs,
- Ne sera pas effective si une des lignes transgresse une contrainte d'intégrité.

L'activation peut se faire dans une commande de création ou de modification de la structure de table :

```
Create Table nom de table ou Alter Table nom de table
( ... ) [ ( ... ) ]
ENABLE { { UNIQUE (colonne, )
          PRIMARY KEY [ CASCADE ]
          CONSTRAINT nomcontrainte }
          [ USING INDEX [ ...
                      TABLESPACE nomtablespace
                      ]
          [ EXCEPTIONS INTO nomtable ]
          ALL TRIGGERS }
;
```

Remarques :

- La clause EXCEPTIONS permet de récupérer l'adresse des lignes qui transgressent les contraintes. La structure de la table des rejets est la suivante :
ROW_ID : ROWID
OWNER : VARCHAR2(30)
TABLE_NAME : VARCHAR2(30)
CONSTRAINT : VARCHAR2(30)
- Activer les contraintes PRIMARY KEY et UNIQUE
 - **construit les index** associés,
 - prend du temps.

6.4.3.2. Inhibées les contraintes ne seront pas contrôlées

- Performance accrue lors de gros chargement de données,
- Automatiquement fait par Sql*Loader.

La désactivation peut se faire dans une commande de création ou de modification de la structure de table :

```
Create Table nom de table ou Alter Table nom de table
( ... ) [ ( ... ) ]
DISABLE { { UNIQUE (colonne, )
          PRIMARY KEY
          CONSTRAINT nomcontrainte }
          [CASCADE]
          ALL TRIGGERS }
;
```

Remarques :

- L'option Cascade permet d'inhiber toutes les dépendances associées. Pour désactiver une clé primaire ou unique utilisée dans une contrainte référentielle, il faut désactiver les clés étrangères en utilisant l'option Cascade.
- Inhiber les contraintes Primary Key et Unique :
 - implique que la contrainte Foreign Key a été inhibée ou que la clause ON DELETE CASCADE a été utilisée,
 - **supprime les index** associés.

7. Mise à jour de la classification

À la vue de ce qui a été dit jusqu'ici, nous sommes en mesure de proposer une nouvelle synthèse des modèles existants :

niveau	données	objets	traitements	flux	
conceptuel	MCD	CVO	MCT	MCF	← MC + MD
analytique	MCDA		MCTA		
organisationnel	MOD		MOT	MOF	→ DOT
logique	MRD		DLT	-	
physique	DPD		-	-	

MCDA : Modèle Conceptuel des Données Analytique

Le MCD accompagné des extensions proposées par MERISE s'appelle MCDA

MCTA : Modèle Conceptuel des Traitements Analytique

Ce modèle assure la cohérence entre le modèle des données et le modèle des traitements. Il tend à être remplacé par le MOT, qui prend en charge également les opérations (consultation, création, modification, suppression) sur les données. De plus le MOT autorise la modélisation de l'organisation du système.

CVO : Cycle de Vie des Objets

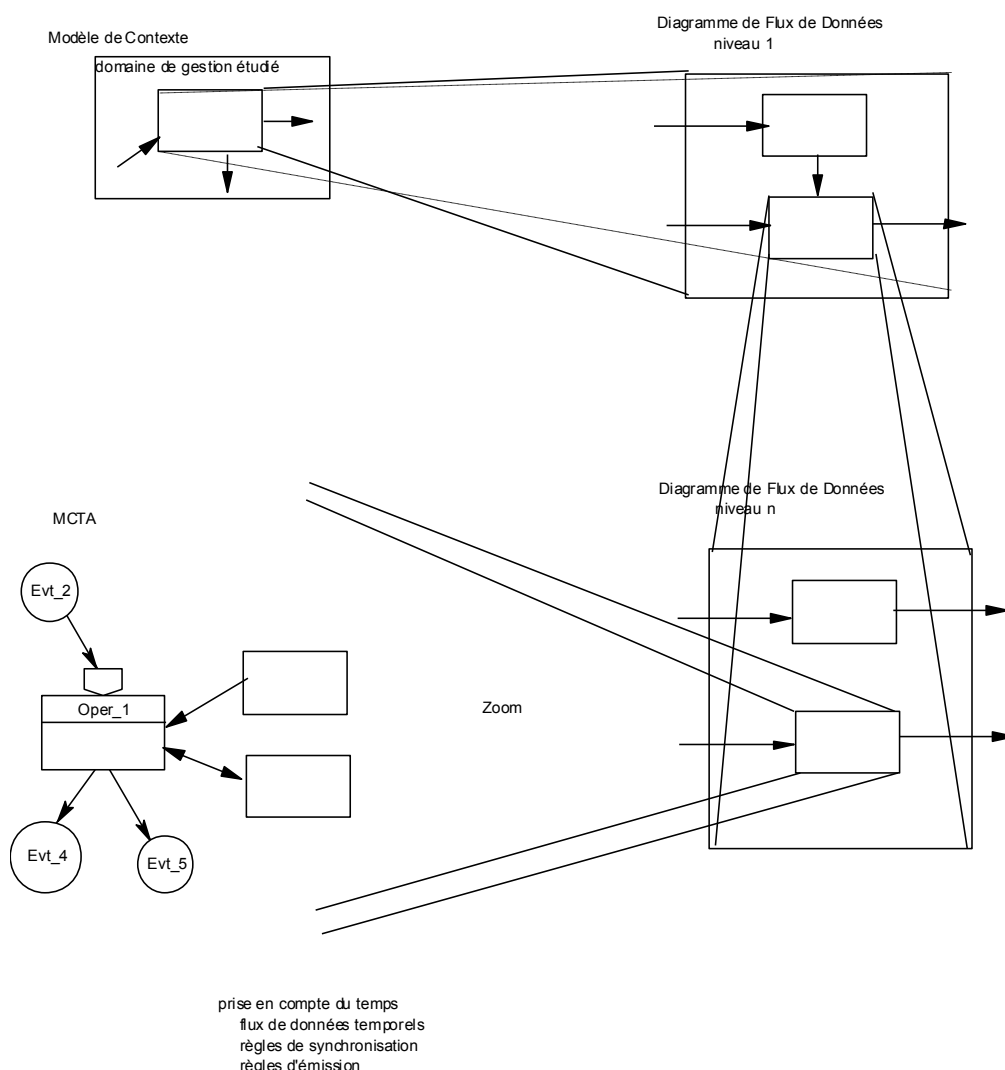
Il est basé sur le MCD. Chaque entité du MCD passe par divers évènements qui traduisent des états de l'objet. Les évènements sont décrits dans les modèles de traitement. Un évènement peut être accompagné d'une alternative ou d'un branchement (boucle ou itération).

DOT : Description Opérationnelle des Traitements

Son but est de décrire l'architecture des logiciels qui devront être réalisés à partir du MOT et la DLT. Les phases prendront l'appellation de phases temps réel (transactions) ou temps différé (batch).

A. Lien diagramme de flux/modèle de traitement

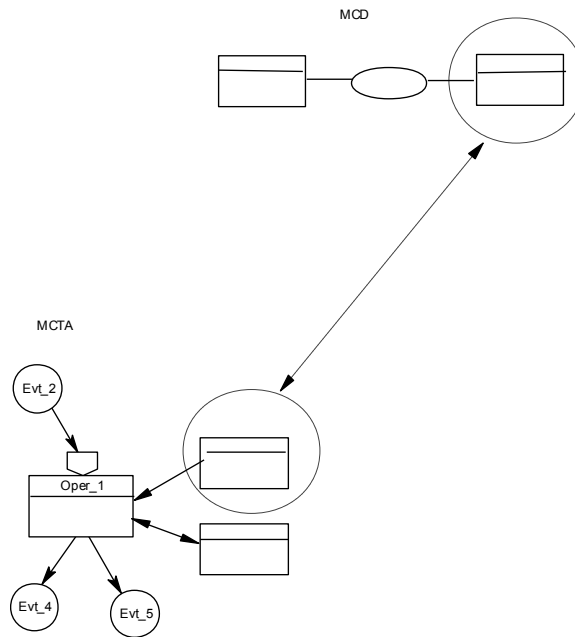
Le modèle de traitement résulte d'un « zoom » sur un diagramme de flux de données de dernier niveau.



B. Lien modèle de données/modèle de traitement

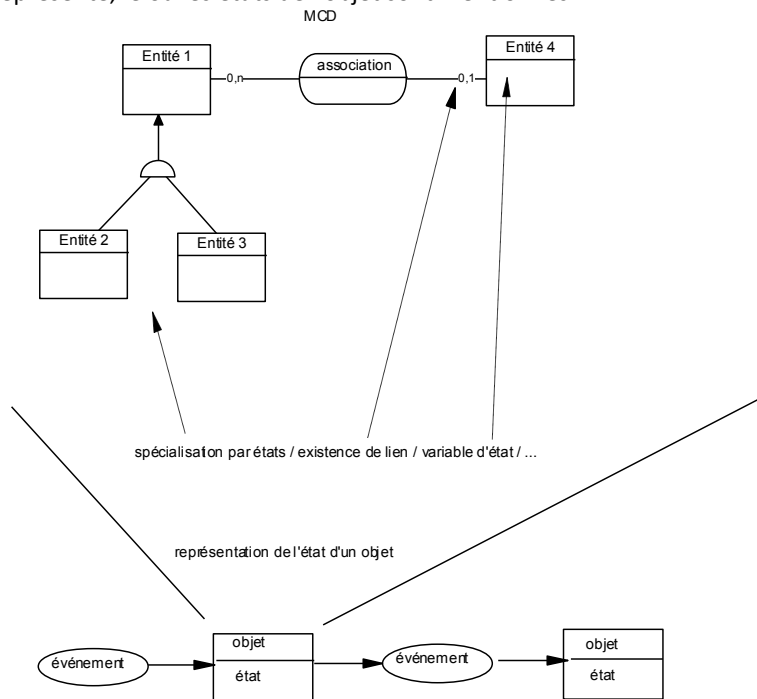
Les données consultées ou mises à jour par une opération dans le modèle de traitement sont représentées graphiquement (client, commande,...).

Rappel : Le MOT par l'intermédiaire d'une phase, permet de faire la même chose.



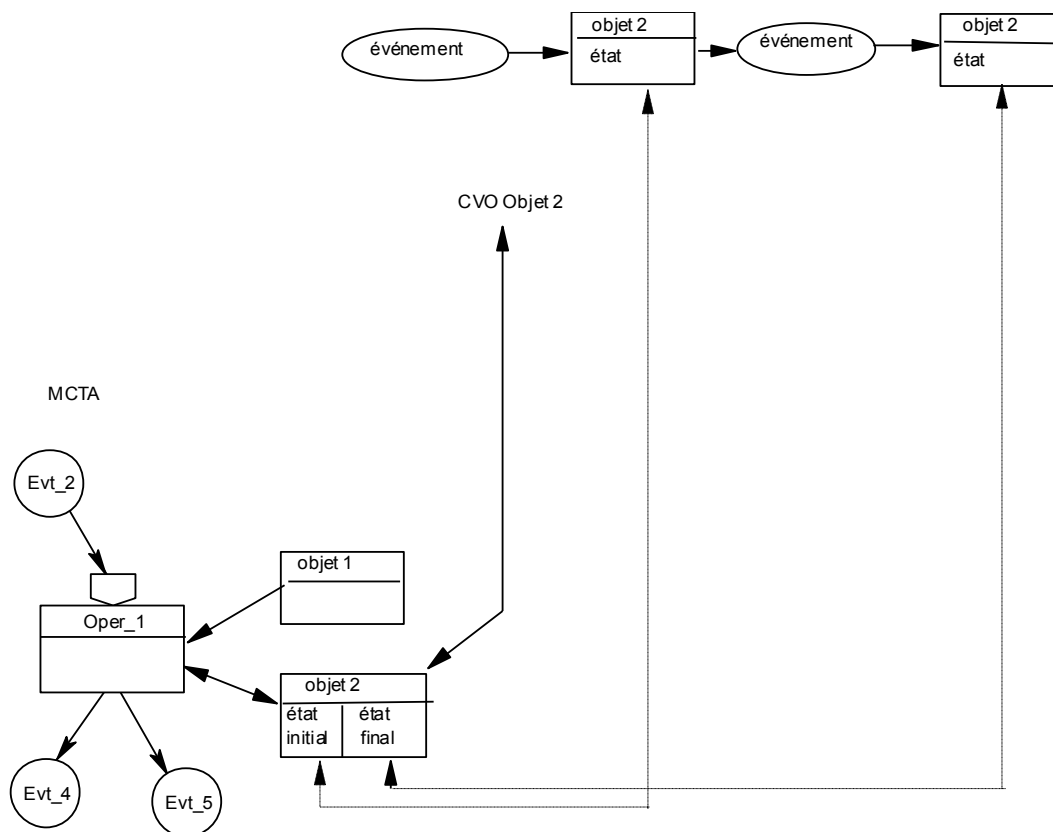
C. Lien modèle de données/cycle de vie des objets

Pour chaque objet représenté, le ou les états de l'objet sont mentionnés :



D. Lien modèle de traitement/cycle de vie des objets

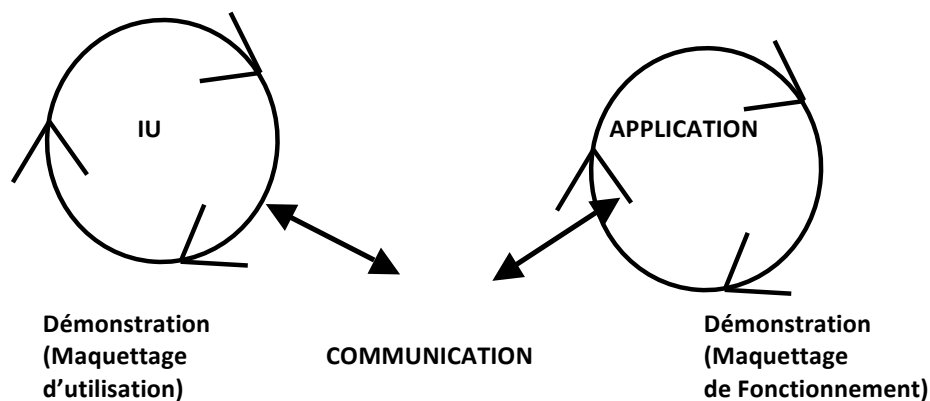
La vie de tout objet est marquée par des événements qui le créent et le font passer par différents stades jusqu'à un état final. Ces différents événements doivent être pris en compte ; différentes actions doivent donc permettre la création de l'objet et les changements d'états qu'il doit subir au cours de sa vie : la validation du modèle de traitement se réalise à l'aide du cycle de vie des objets.



8. Axe de modélisation avec MERISE/2

8.1. Introduction

La méthode MERISE/2 propose de compléter la conception de la logique de fonctionnement de l'application par une démarche privilégiant le point de vue de l'utilisateur pour favoriser l'ergonomie, la convivialité et la souplesse d'utilisation de ces applications. Cela passe par une démonstration régulière exploitant au maximum les possibilités de maquettage actuelles dans le cadre d'une démarche itérative (cf schéma ci dessous).



Ces outils de représentation, utilisés par les professionnels de l'informatique, sont aussi des outils de communication sur l'organisation des activités de l'entreprise. Ils contribuent à rapprocher le monde de la gestion et le monde de l'informatique.

Cette partie communication s'avère efficace lorsque l'utilisateur comprend les modèles présentés par le concepteur de l'application.

Nous allons retrouver comme principaux modèles :

- Le Modèle de Flux
- Le Modèle de Données
- Le Modèle de Traitement

Ces modèles sont basés sur 3 axes de modélisation.

8.2. Les trois axes de modélisation

- L'axe architecture (fonctionnelle) du système d'information qui permet de décrire ce que fait le système d'information.
- L'axe statique du système d'information (SI) qui permet de décrire ce qu'est le système, c'est-à-dire l'organisation des données.
- L'axe cinématique qui permet de décrire comment se comporte le système, c'est-à-dire les traitements et la succession des transformations qu'ils opèrent sur les données.

Le sens de la rénovation

On évolue de la représentation uniquement statique du système d'information de l'entreprise (le MCD aujourd'hui bien maîtrisé), vers un ensemble de représentations qui, en outre, prennent en compte les transformations opérées sur les données par des traitements et la définition de l'architecture du système d'information.

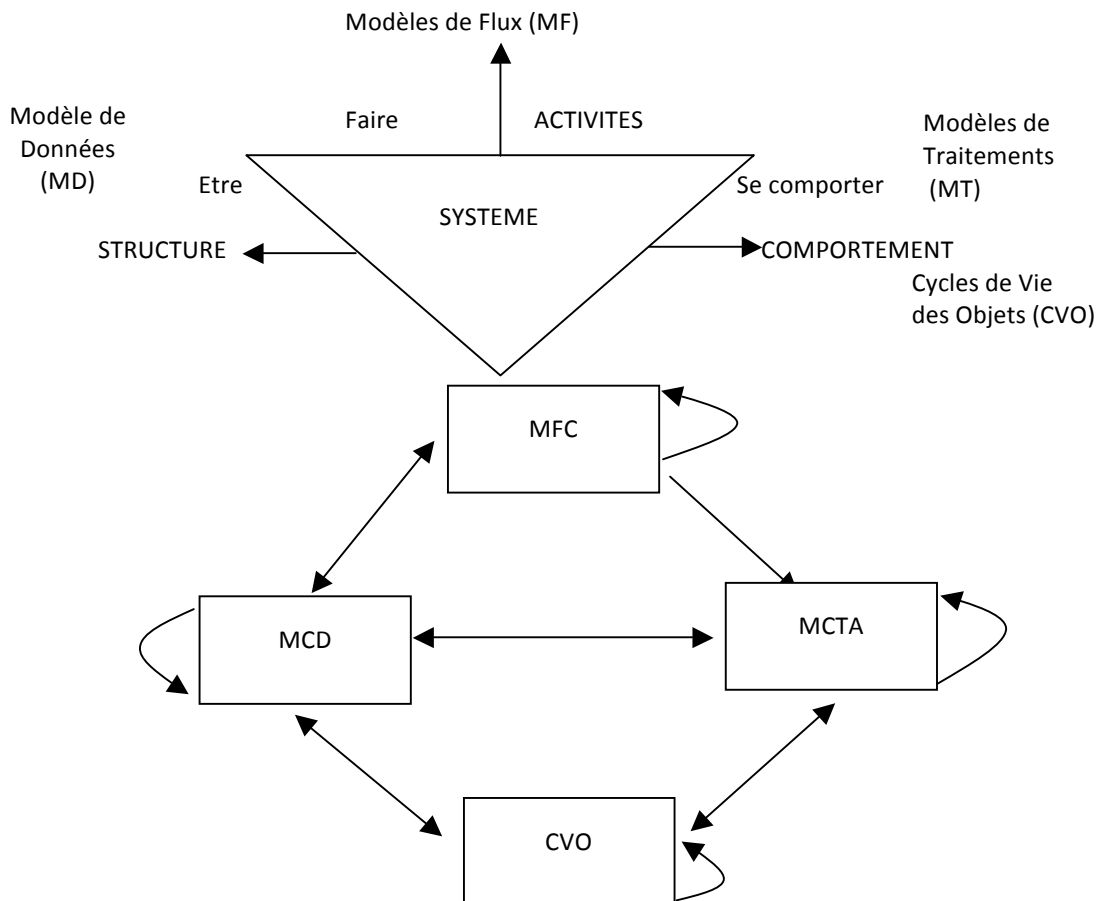
S Y S T E M E D' I N F O R M A T I O N	ARCHITECTURE	MF Modèle de Flux	L'application "fait"	Cette partie décrit les activités du système, les liens entre les différents acteurs.
	STATIQUE	MD Modèle de Données	L'application « est »	Cette partie décrit la structure du système, de quoi il est composé.
	DYNAMIQUE	MT : Modèle de Traitement CVO : Cycle de Vie des Objets	L'application "se comporte"	Cette partie décrit le comportement du système, les différents états qu'il peut prendre.

Les Modèles de Flux représentent des mouvements d'informations à l'intérieur d'un domaine d'étude et entre ce domaine et le monde extérieur. Ils permettent de rechercher les activités et mettent l'accent sur leurs interactions avec l'environnement.

Le Modèle des Données prend en compte les données liées à l'organisation, leur durée de vie, le système d'autorisation d'utilisation des données, et distingue les données manuelles et les données informatisées. Il intéresse le gestionnaire par le fait qu'il visualise parfaitement quelles sont les données manipulées.

Le Modèle des Traitements représente les traitements effectués sur les données, formalise les actions possibles sur les données, sur un site ou un poste donné. En somme, il représente très concrètement **qui** (le gestionnaire), **fait quoi** quotidiennement (**quand**), derrière quel poste de travail (**où**).

Ces différents modèles sont élaborés soit sur la totalité du système d'information, soit sur une partie du système d'information. Ils s'inscrivent dans le cadre d'une démarche itérative.



On voit bien apparaître ici le fait que la démarche est **itérative**, les flèches représentent le fait que la conception d'un modèle permet de valider celui-ci dans un premier temps. Dans un deuxième temps ce nouveau modèle élaboré permet de valider les précédents modèles conçus.

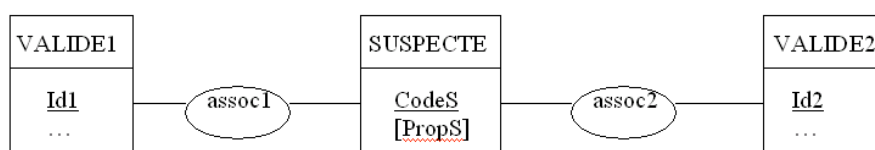
9. Florilège de Pratiques recommandées (☺), pratiques (☹) ou douteuses (⊗)

A. Justification des entités suspectes (☺)

Une entité suspecte est une entité réduite à un identifiant ou comportant une seule propriété à laquelle on a ajouté un identifiant artificiel (du genre « code de... »).

Comment justifier leur maintien ?

1. Elle participe à au moins deux associations valides : c'est une **charnière sémantique**.

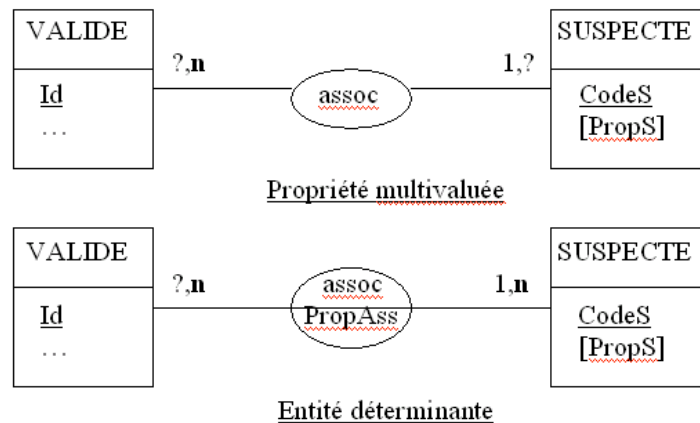


2. Elle ne participe qu'à une seule association mais :

a. Son rôle est doté d'une cardinalité minimale égale à 0 : c'est une **entité autonome**



b. Le rôle de l'autre entité est doté d'une cardinalité maximale égale à n : c'est une **propriété multivaluée** ou une **entité déterminante**.



B. L'entité DATE

La prise en compte du temps dans le modèle des données n'est pas toujours aisée. De fait, les problèmes de planification et « d'historisation » font partie des problèmes les plus délicats à résoudre. C'est sans doute ce qui justifie que, dans notre milieu, on fasse un emploi fréquent de l'entité DATE. Fréquent mais trop souvent inconsideré. Dans la réalité des systèmes de gestion, il est extrêmement rare qu'un concept soit identifiable à l'aide de l'intervention d'une mesure du temps. C'est pourquoi, **il ne nous semble pas pertinent de faire reposer l'évaluation des compétences en matière de modélisation sur l'intervention d'une entité DATE.**

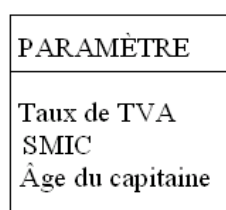
Tâchons d'en user avec parcimonie, quand le sujet nous l'impose, notamment, et avec cohérence : comment comprendre que, dans un même schéma, il y ait des dates entités et des dates propriétés ? **Si les dates sont à considérer comme classe de référence, nous en ferons une entité.**

C. L'entité PARAMÈTRE (⊗⊗⊗)

Certains auteurs utilisent une entité PARAMÈTRE qui, par définition, n'est reliée à aucune autre entité.

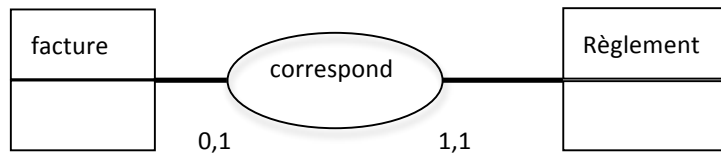
Il s'agit d'une espèce de paramétrage des constantes du système comme on peut le faire en programmation (exemple : un taux unique de TVA, constant... pendant un certain temps).

Le principe n'est pas critiquable. Il n'en va pas de même de la représentation : regrouper en une entité des informations aussi diverses que variées ne fait que brouiller la compréhension de la notion d'entité (regroupement décrivant une réalité perceptible, identification, instanciation d'occurrences, etc.). Nous ne choisirons pas cette solution et conseillons, si le besoin s'en fait sentir, de mentionner les constantes du système en une **contrainte d'intégrité hors schéma.**

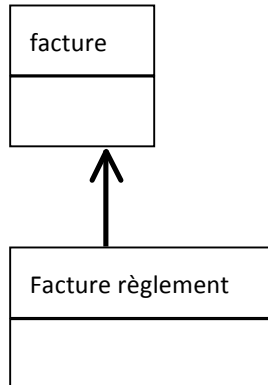


D. Masquage d'une spécialisation (/!\)(☹)

Les relations du type 0,1 – 1,1 peuvent cacher une spécialisation. Dans ce cas, il n'est pas nécessaire de maintenir la relation, un héritage peut remplacer la relation.



La modification à réaliser est la suivante :



E. Relations binaires 0,1-0,1 et 1,1-1,1

Ce genre de relation peut se rencontrer. Pour une association 0,1-0,1 placer la clé étrangère là où cela sera le mieux.

Une association 1,1-1,1 se traduit par une relation qui dispose de deux clés candidates. On peut également faire passer les clés étrangères de deux côtés.

Rappel : Une clé candidate est un attribut qui n'est pas clé primaire mais qui est apte à assurer l'unicité d'une ligne. Aucune représentation n'est standardisée.

10. Les logiciels de modélisation (AGL ateliers de génie logiciel)

Plusieurs produits sont présents sur le marché :

Les vingt-deux AGL recensés par l'Adeli						
AGL	Editeur	Méthodes structurées supportées	Méthodes objet supportées	Coopération des méthodes	Plates-formes	Prix pour 1 poste (KF)
AGL-X	Prisme	Merise, SA	-	-	Windows, AS/400	50
AMC*Designer	Sybase France	Merise	OMT, autres	X	Windows	5,2
Conceptor	Telis Cegelog	Merise, autres	-	-	Windows	9,8
Excelsior II	Intersolv	IE, SA	OMT, OOSE, autres	X	Windows, OS/2	26
GraphTalk OMT	Continuum SOCS	-	OMT, UML	X	Windows	25
Key:Enterprise	Sterling Software	IE	-	X	OS/2	nc
Key:WorkGroup	Sterling Software	IE, autres	OMT, UML	X	Windows	nc
Mega - ISOA	Mega International	Merise, IE, autres	OOSE, UML	X	Windows, OS/2	15
Merise/Maestro II	Softlab	Merise	-	-	Windows, OS/2	55
Objecteering	Softteam	-	OMT, UML, Classe-Relation	X	Windows	15
ObjectPool	Teamsa Informatique	autres	autres	-	Windows, Unix, AS/400	40
Oracle Designer/2000	Oracle France	autres	-	-	Windows, OS/2	28
Pacdesign	CGI Informatique	Merise, autres	autres	-	Windows, OS/2	61
Paradigm Plus	Platinum Technology	-	OMT, UML, OOAD, autres	-	Windows, OS/2, Unix	24
Principia	Sema Group	Merise	-	-	Windows	50
Rational Rose	Rational Software	-	OMT, UML, OOAD, OOSE	X	Windows, Unix	17,3
Select/Enabler	Softlab	autres	OMT, UML, OOSE	X	Windows, OS/2	50,5
Select/Entreprise	Microdata Soft	Merise, non	OMT, UML, autres	X	Windows	21,5
Silverrun	CSA France	Merise, IE, autres	OMT, UML, Classe-Relation	X	Windows, OS/2, Unix, Macintosh	45
Système Delf	Delf	Merise	-	-	Windows, OS/2	25
Tramis	Concis Technologies	Merise, IE, SA	OMT, UML, OOM, autres	X	Windows	18,3
Win'Design	Cecima	Merise	-	-	Windows	16,9

Remarque : La plus grande partie des produits tourne exclusivement sous Windows !

11. les exercices (pour la forme !)

Ce cours est accompagné d'un support comportant des exercices d'application.