Conception d'une base de données

Cyril Gruau*

13 novembre 2003

Résumé

Ce support de cours regroupe quelques notions concernant le modèle entité-association, le schéma relationnel et la traduction de l'un vers l'autre.

Mots-clef : Merise, MCD, entité, association, MLD, relation, traduction, MPD

Table des matières

*Cyril.Gruau@ensmp.fr

Introduction

Les techniques présentées ici font partie de la méthodologie Merise élaborée en France en 1978 (cf. [5]) qui permet notamment de concevoir un système d'information d'une façon standardisée et méthodique.

Le but de ce support de cours est d'introduire le schéma entité-association, le schéma relationnel et d'expliquer la traduction entre les deux.

Ne sont pas abordés ici : les dépendances fonctionnelles, les contraintes, les traitements, les langages relationnels, la gestion de projet. Pour tout cela, le lecteur est dirigé vers [2, 3]. La modélisation objet ne fait pas non plus partie des outils exposés dans ce document.

1 Modèle conceptuel de données (MCD)

Avant de réfléchir au schéma relationnel d'une application, il est bon de modéliser la problématique à traiter d'un point de vue conceptuel et indépendamment du logiciel utilisé. C'est le but de cette partie.

1.1 Schéma entité-association

Une *entité* est une population d'individus n'ayant que des caractéristiques comparables.

Par exemple, dans une entreprise qui achète et vend des produits, on a l'entité `clients`, l'entité `articles` et l'entité `fournisseurs`.

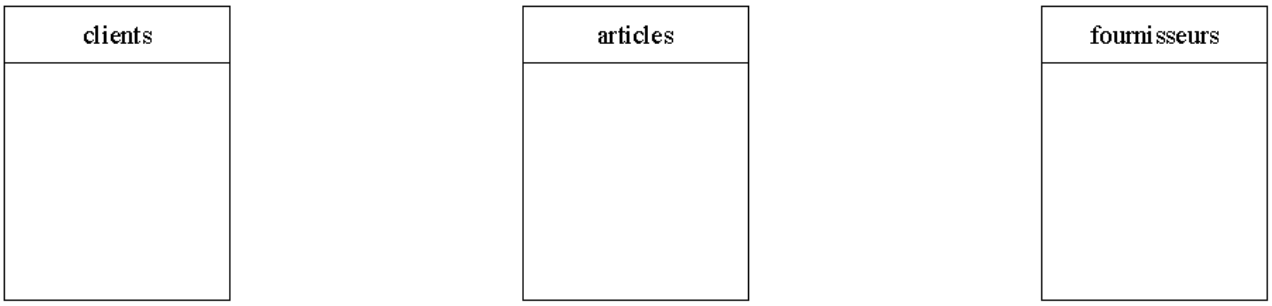

FIG. 1 – *Entités*

Une *association* est un lien entre plusieurs entités.

Dans notre exemple, l'association `acheter` fait le lien entre les entités `articles` et `clients`, tandis que l'association `livrer` fait le lien entre les entités `articles` et `fournisseurs`.

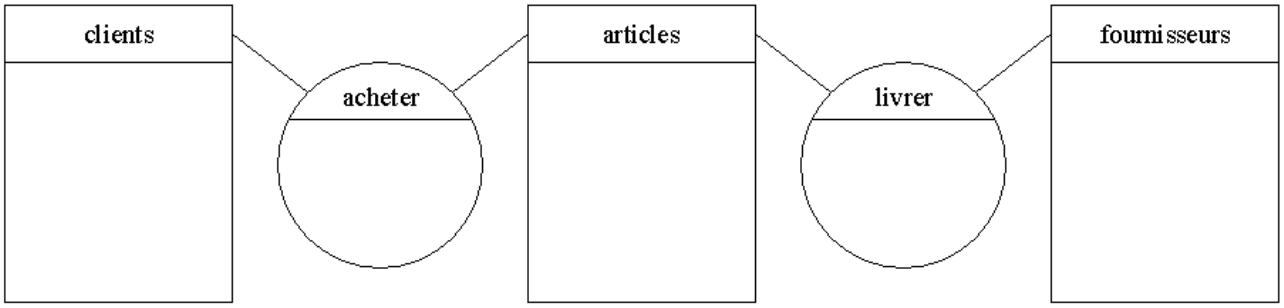

FIG. 2 – *Associations*

Un *attribut* est une propriété d'une entité ou d'une association.

Toujours dans notre exemple, le `prix unitaire` est un attribut de l'entité `articles`, le `nom du client` est un attribut de l'entité `clients`. La `quantité commandée` est un attribut de l'association `acheter`, la `date de livraison` est un attribut de l'association `livrer`.

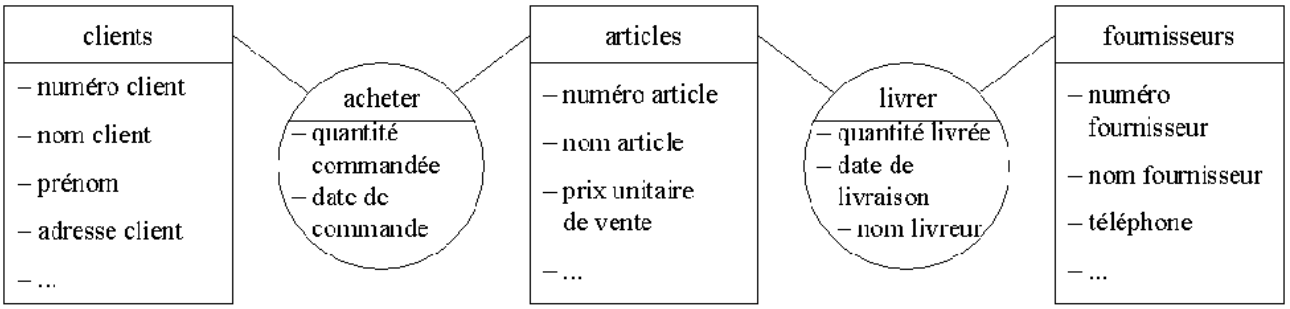

FIG. 3 – *Attributs*

Chaque individu d'une entité doit être identifiable de manière unique. C'est pourquoi les entités doivent posséder un attribut sans doublon, l'*identifiant*.

Le numéro de client est l'identifiant de l'entité `clients` (on souligne cet attribut sur le schéma).

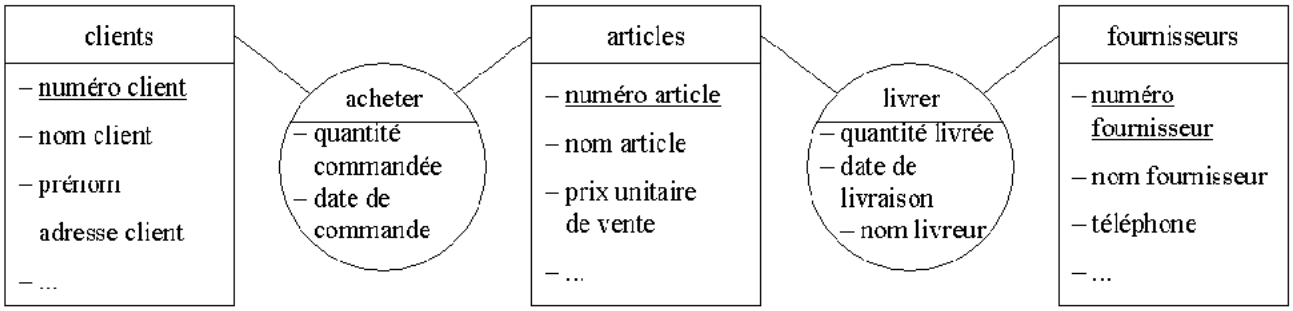

FIG. 4 – *Identifiants*

Remarques :
- un attribut ne doit pas figurer dans deux entités ou associations différentes (donc il faut spécialiser l'attribut `nom` en `nom du client`, `nom du produit` et `nom du fournisseur`) ;
- une entité possède au moins un attribut (son identifiant) ;
- une association peut ne pas posséder d'attribut.

Conseils :
- éviter les identifiants composées de plusieurs attributs (comme par exemple un identifiant formé par les attributs `nom du client` et `prénom`) ;
- il faut un identifiant totalement indépendant des autres attributs (éviter par exemple d'ajouter un identifiant `NomPrénom` qui serait la concatenation des attributs `nom du client` et `prénom`) ;
- préférer un identifiant court pour rendre la recherche la plus rapide possible (éviter par exemple les chaînes de caractères comme le numéro de sécurité sociale ou la plaque d'immatriculation) ;
- conclusion : dans le modèle physique de données (cf. 3), on utilise une clé numérique (un entier) incrémentée automatiquement.

La *cardinalité* d'un lien entre une entité et une association est le minimum et le maximum de fois qu'un individu de l'entité peut être concerné par l'association.

Un client a au moins acheté un article et peut acheter `n` articles (`n` étant indéterminé), tandis qu'un article peut avoir été acheté entre `0` et `n` fois (même si ce n'est pas le même `n`) et n'est livré que par `1` fournisseur. On obtient alors le schéma entité-association complet suivant :

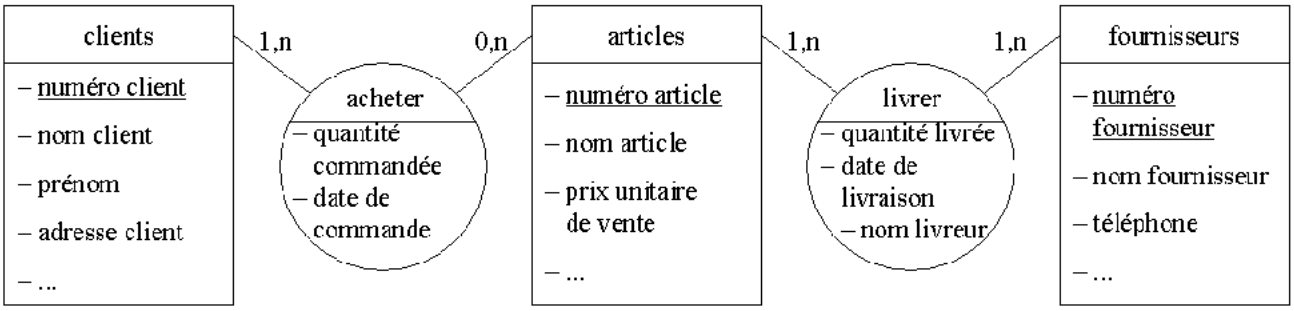

FIG. 5 – *Cardinalités*

Remarque : si une cardinalité est connue et vaut 2, 3, etc. on considère quand même qu'elle est indéterminée `n`, de telle sorte qu'on ne puisse avoir que des cardinalités 0, 1 ou `n`. Cela se justifie par le fait que même si on connaît `n` au moment de la conception, il se peut que cette valeur évolue avec la politique de l'entreprise. Il vaut donc mieux considérer `n` comme une inconnue dès le départ.

1.2 Cas particuliers

Exemple d'association binaire *réflexive* : dans ce cas, un employé est dirigé par un employé (sauf le

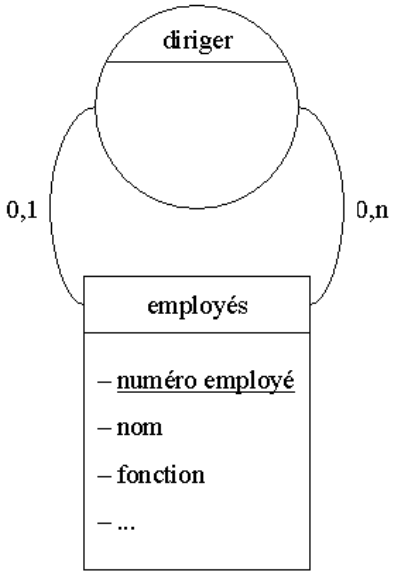

FIG. 6 – *Association réflexive*

directeur général) et un employé peut diriger plusieurs employés.

Exemple d'association entre trois entités (association *ternaire*) : dans ce cas, d'une part un vol peut

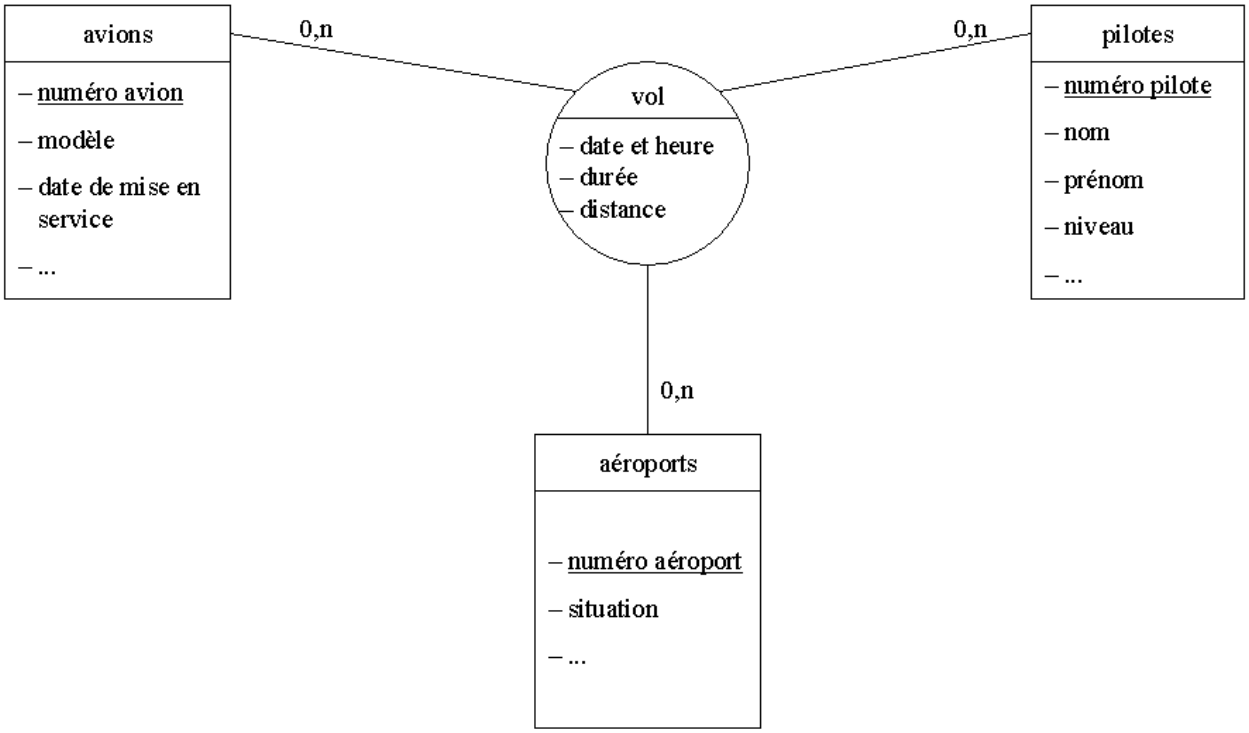

FIG. 7 – *Association ternaire*

concerner plusieurs avions (si c'est un vol avec escale), plusieurs pilotes (s'il y a un co-pilote) et plusieurs aéroports (départ/escales/arrivée) d'où la nécessité d'une association ternaire et d'autre part, un avion, un pilote ou un aéroport peuvent être utilisés 0 ou plusieurs fois par l'ensemble des vols (d'où les cardinalités).

Exemple de plusieurs associations entre deux mêmes entités : dans ce cas, un être humain peut être

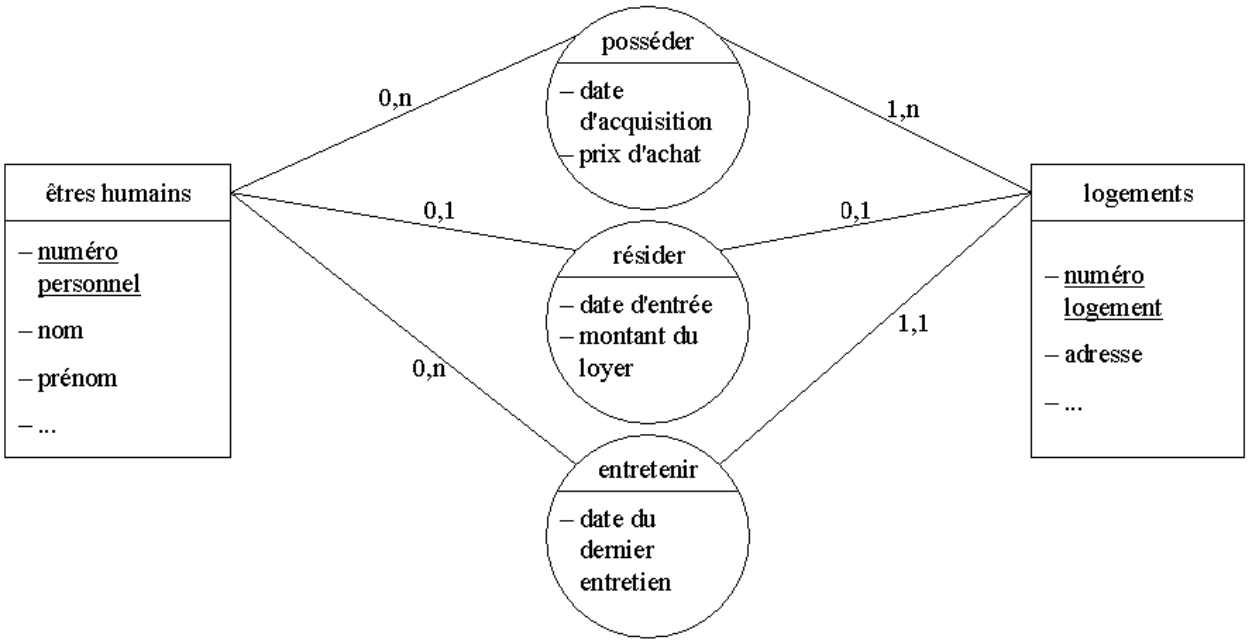

FIG. 8 – *Associations plurielles*

propriétaire, locataire et/ou chargé de l'entretien. On suppose qu'un être humain ne réside que dans un logement au maximum, qu'un logement n'est occupé que par une personne au maximum et qu'un logement est entretenu par une et une seule personne (il s'agit d'un exemple).

1.3 Règles de normalisation

Première forme normale : chaque entité doit posséder un identifiant qui caractérise ses individus de manière unique.

Deuxième forme normale : l'identifiant peut être composée de plusieurs attributs mais les autres attributs de l'entité doivent être dépendant de l'identifiant en entier (et non pas une partie de cet identifiant).

Ces deux premières formes normales peuvent être oubliée si on suit le conseil de n'utiliser que des identifiants non composés de type numéro.

Troisième forme normale (importante) : les attributs d'une entité doivent dépendre directement de son identifiant.

Par exemple, la date de fête d'un client ne dépend pas de son identifiant `numéro de client` mais plutôt de son `prénom`. Elle ne doit pas figurer dans l'entité `clients`, il faut donc faire une entité `calendrier` à part, en association avec `clients`.

Forme normale de Boyce-Codd (à oublier également) : les attributs d'un identifiant composé ne doivent pas dépendre d'un autre attribut de l'entité.

À ce stade, seules les entités ont été normalisées, il reste les associations.

Normalisation des relations[1] (importante) : les attributs des associations doivent dépendre des identifiants de toutes les entités en association.

Par exemple, la `quantité commandée` dépend à la fois du `numéro de client` et du `numéro d'article`, par contre la `date de commande` non. Il faut donc faire une entité `commandes` à part :

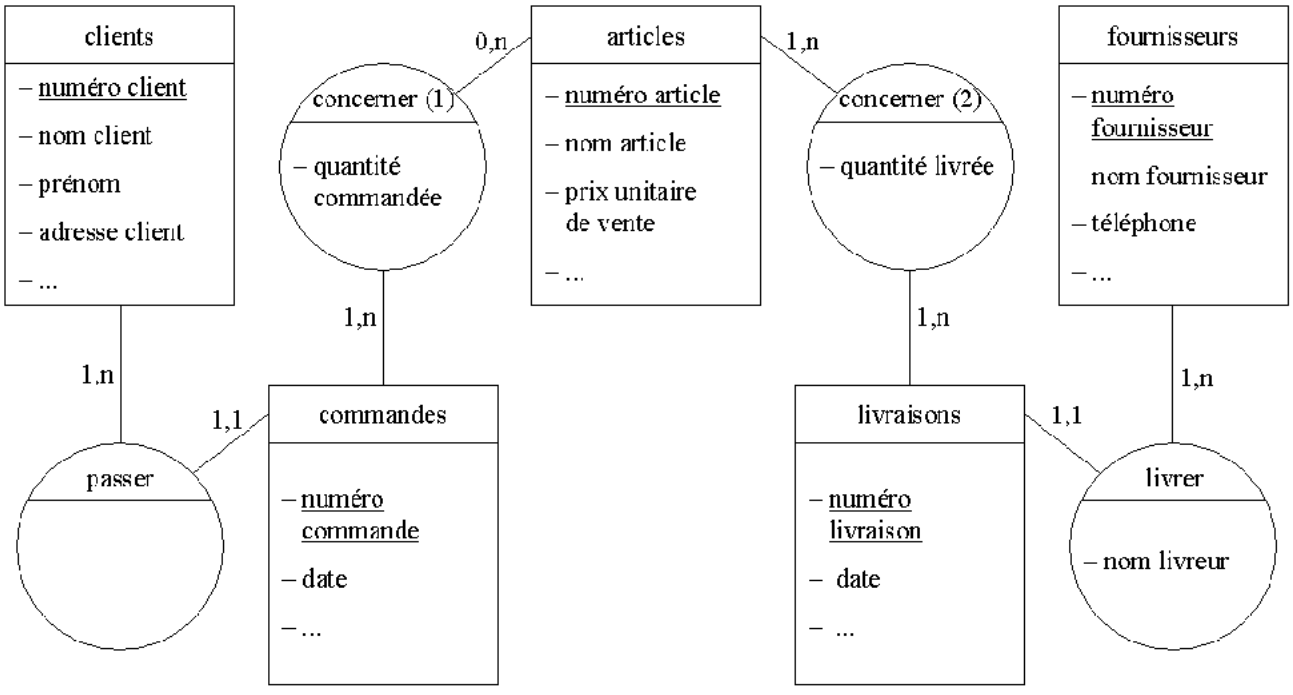

FIG. 9 – *Normalisation des relations*

1. comprendre : normalisation des associations

1.4 Méthodologie

Face à un problème bien formulé (même si ça n'existe pas) procéder ainsi :

- identifier les entités en présence ;
- lister leurs attributs ;
- ajouter les identifiants ;

- établir les associations entre les entités ;
- lister leurs attributs ;

- éliminer les *synonymes* (plusieurs signifiants pour un signifié) et les *polysèmes* (plusieurs signifiés pour un signifiant) ;
- calculer les cardinalités ;

- vérifier la troisième forme normale et la normalisation des relations (surtout pour les associations non binaires) ;
- effectuer les corrections nécessaires.

Il faut garder également à l'esprit que le modèle doit être *exhaustif* (c'est-à-dire contenir toutes les informations nécessaires) et éviter toute *redondance* (le prix unitaire d'un article n'a pas besoin de figurer dans l'entité `commandes`, c'est la troisième forme normale).

2 Modèle logique de données (MLD)

Maintenant que le MCD est établit, on peut le traduire en différents systèmes logiques, comme les systèmes par fichiers ou les bases de données.

2.1 Systèmes logiques

Avant l'apparition des systèmes de gestion de base de données (SGBD ou DBMS pour Data Base Management System en anglais), les données étaient stockées dans des fichiers binaires et gérées par des programmes exécutables (Basic, Cobol ou Dbase par exemple). La maintenance des programmes (en cas de modification de la structure des données par exemple) était très problématique. Voir [1] pour une traduction d'un MPD vers un MLD fichier.

Sont alors apparus les SGBD *hiérarchiques* dans lesquels les données sont organisées en arbre (IMS-DL1 d'IBM par exemple), puis les SGBD *réseaux* dans lesquels les données sont organisées selon un graphe plus général (IDS2 de Bull par exemple). Voir [2, 3, 1] pour une traduction d'un MPD vers un MLD Codasyl (base de données réseaux). Ces deux types de SGBD sont dit *navigationnels* car on peut retrouver l'information à condition d'en connaître le chemin d'accès.

Aujourd'hui, ils sont largement remplacés par les SGBD relationnels (SGBDR) avec lesquels l'information peut être obtenue par une requête formulée dans un langage quasiment naturel. Ce sont les SGBD les plus répandus (Oracle et DB2 d'IBM par exemple). Nous nous contentons donc ici du modèle logique de données relationnel (MLDR).

Plus récemment, sont apparus des SGBD *orientés objets* qui offrent à la fois un langage de requête et une navigation hypertexte. Un modèle logique de données orienté objet permet par exemple de concevoir des classes Java s'appuyant sur une base de données relationnelle et permettant le développement d'une application web.

2.2 Schéma relationnel

Concentrons-nous sur le MLDR. Lorsque des données ont la même structure (comme par exemple, les bordereaux de livraison), on peut les organiser en *table* dans laquelle les colonnes décrivent les champs en commun et les lignes contiennent les valeurs de ces champs pour chaque enregistrement.

Les lignes d'une table doivent être uniques, cela signifie qu'une colonne (au moins) doit servir de *clé primaire*. La clé primaire d'une ligne ne doit pas changer au cours du temps et ne peut contenir la valeur `NULL`, alors que les autres colonnes le peuvent.

Par ailleurs, il se peut qu'une colonne `Colonne1` d'une table ne doive contenir que des valeurs prises par une colonne `Colonne2` d'une autre table (par exemple, le numéro du client sur une commande doit correspondre à un vrai numéro de client). La `Colonne2` doit être sans doublons (bien souvent il s'agit d'une clé primaire) et on dit que la `Colonne1` est *clé étrangère*.

Par convention, on souligne les clés primaires et on fait précéder les clés étrangères d'un dièse `#` dans la description des colonnes d'une table :

```
clients(numéro client, nom client, prénom, adresse client, ...)
commandes(numéro commande, date, #numéro client, ...)
```

Remarques :
- une même table peut avoir plusieurs clés étrangères mais une seule clé primaire (éventuellement composées de plusieurs colonnes) ;
- une clé étrangère peut aussi être primaire ;
- une clé étrangère peut être composée (c'est le cas si la clé primaire en liaison est composée).

La section suivante contient des exemples.

On peut représenter les tables d'une base de données relationnelle par un schéma relationnel dans lequel les tables sont appelées *relations* et les liens entre les clés étrangères et leur clé primaire est symbolisé par un connecteur :

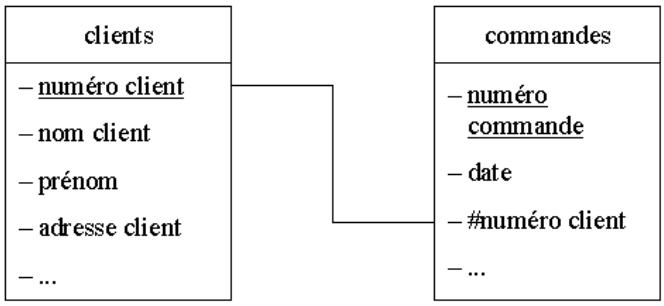

FIG. 10 – *Schéma relationnel*

2.3 Traduction

Pour traduire un MCD en troisième forme normale en un MLDR, il suffit d'appliquer cinq règles (à connaître par cœur). Mais avant, on dit qu'une association entre deux entités (éventuellement réflexive) est de type :
- 1 : 1 si les deux cardinalités sont 0,1 ou 1,1 ;
- 1 : n si une des deux cardinalité est 0,n ou 1,n ;
- n : m (plusieurs à plusieurs) si les deux cardinalités sont 0,n ou 1,n.

En fait, un schéma relationnel ne peut faire la différence entre 0,n et 1,n. Par contre, il peut la faire entre 0,1 et 1,1 (cf. règles 2 et 3).

Règle 1 : toute entité devient une table dans laquelle les attributs deviennent des colonnes. L'identifiant de l'entité constitue alors la clé primaire de la table.

Par exemple, l'entité `articles` de la figure 9 devient la table :

```
articles(numéro article, nom article, prix unitaire de vente, ...)
```

Règle 2 : dans le cas de deux entités reliées par une association de type 1 : n, on ajoute une clé étrangère dans la table côté 0,1 ou 1,1, vers la clé primaire de la table côté 0,n ou 1,n. Les attributs de l'association glissent vers la table côté 0,1 ou 1,1. Et si la cardinalité est 1,1 alors la clé étrangère ne peut recevoir la valeur `NULL` (autrement dit, vide interdit).

Par exemple, l'association `livrer` de la figure 9 est traduite par :

```
fournisseurs(numéro fournisseur, nom fournisseur, téléphone, ...)
livraisons(numéro livraison, date, #numéro fournisseur (non vide), nom livreur)
```

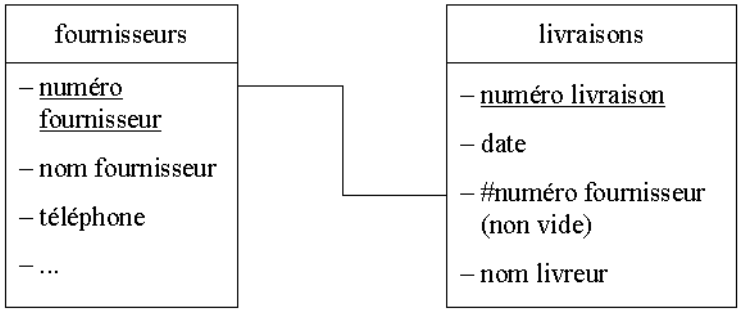

FIG. 11 – *Traduction d'une association de type 1 : n*

Règle 3 : dans le cas de deux entités reliées par une association de type 1 : 1, on ajoute, aux deux tables, une clé étrangère vers la clé primaire de l'autre. Afin d'assurer la cardinalité maximale de 1, on ajoute une contrainte d'unicité sur chaque de ces clés étrangères (la colonne correspondante ne peut prendre que des valeurs distinctes). Les attributs de l'association sont alors repartis vers l'une des deux tables. Et si la cardinalité est 1,1 alors la clé étrangère concernée ne peut recevoir la valeur `NULL` (autrement dit, vide interdit).

Par exemple, l'association `résider` de la figure 8 est traduite par :

```
êtres humains(numéro personnel, nom, prénom, #numéro appartement (unique),
 date d'entrée, ...)
logement(numéro appartement, adresse, #numéro personnel (unique),
 montant du loyer, ...)
```

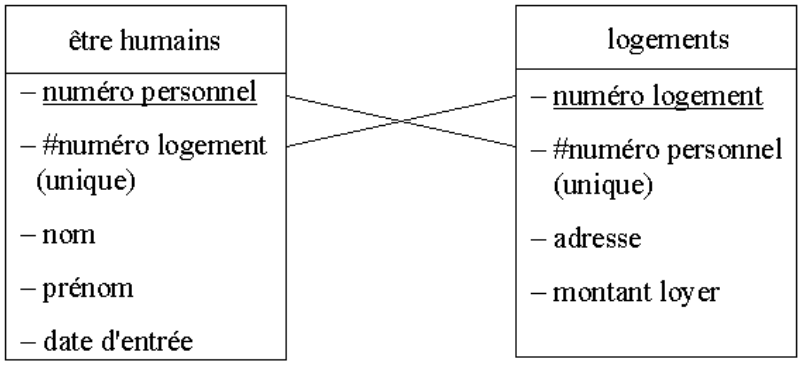

FIG. 12 – *Traduction d'une association de type 1 : 1*

En fait, la règle 3 considère que le type 1 : 1 correspond à deux type 1 : n symétriques. Autre technique : ajouter une table intermédiaire dont la clé primaire est composée de clés étrangères vers les clés primaires des tables en association et une contrainte d'unicité sur ces clés étrangères (c'est-à-dire considérer le type 1 : 1 comme un cas particulier du type n : m) :

```
êtres humains(numéro personnel, nom, prénom, ...)
logement(numéro appartement, adresse, ...)
résider(#numéro personnel (unique), #numéro appartement (unique),
 date d'entrée, montant du loyer)
```

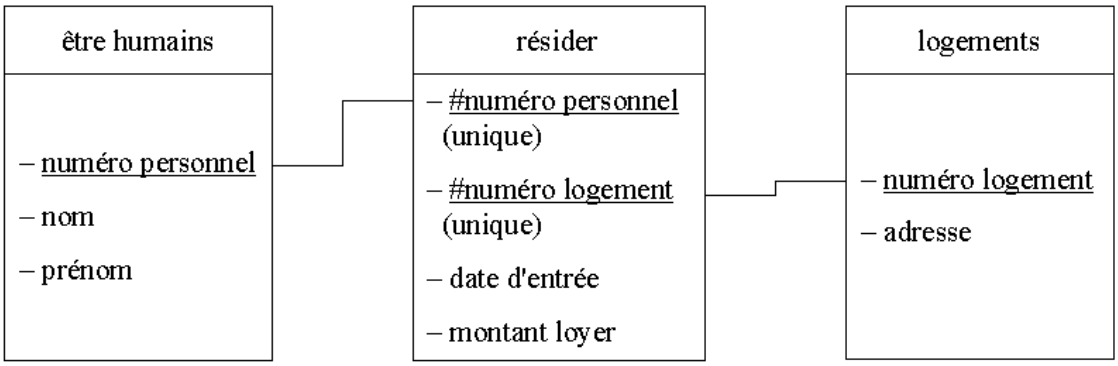

FIG. 13 – *Traduction alternative d'une association de type 1 : 1*

Cette alternative est sans doutes préférable, car plus évolutive (si le type 1 : 1 est un jour converti en un autre type).

Remarque : d'autres techniques sont parfois proposées pour la règle 3 (fusionner les tables, utiliser une clé primaire identique) mais en pratique elles ne sont pas exploitables dans le cas général.

Règle 4 : une association entre deux entités et de type n : m est traduite par une table supplémentaire (parfois appelée *table de jointure*) dont la clé primaire est composée de deux clés étrangères vers les clés primaires des deux tables en association. Les attributs de l'association deviennent des colonnes de cette table.

Par exemple, l'association `concerner (1)` de la figure 9 est traduite par :

```
articles(numéro article, nom article, prix unitaire de vente, ...)
lignes de commande(#numéro commande, #numéro article, quantité commandée)
commandes(numéro commande, date, ...)
```

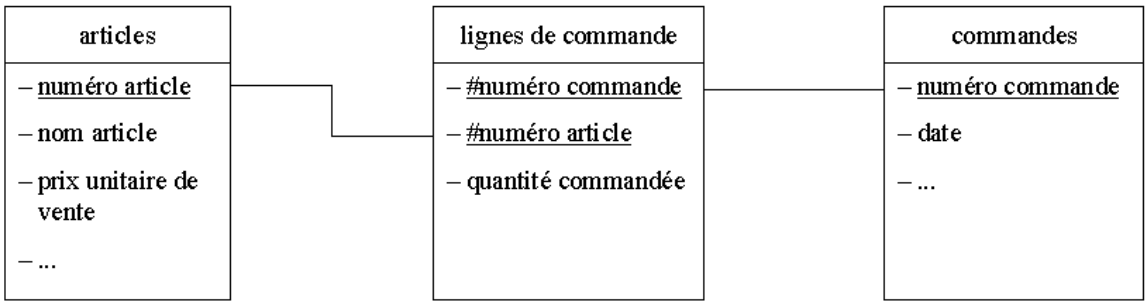

FIG. 14 – *Traduction d'une association de type n : m*

Règle 5 : une association non binaire est traduite par une table supplémentaire dont la clé primaire est composées d'autant de clés étrangères que d'entités. Les attributs de l'association deviennent des colonnes de cette table.

Par exemple, l'association `vols` de la figure 7 devient la table :

```
vols(#numéro avion, #numéro pilote, #numéro aéroport, date et heure, durée, distance)
```

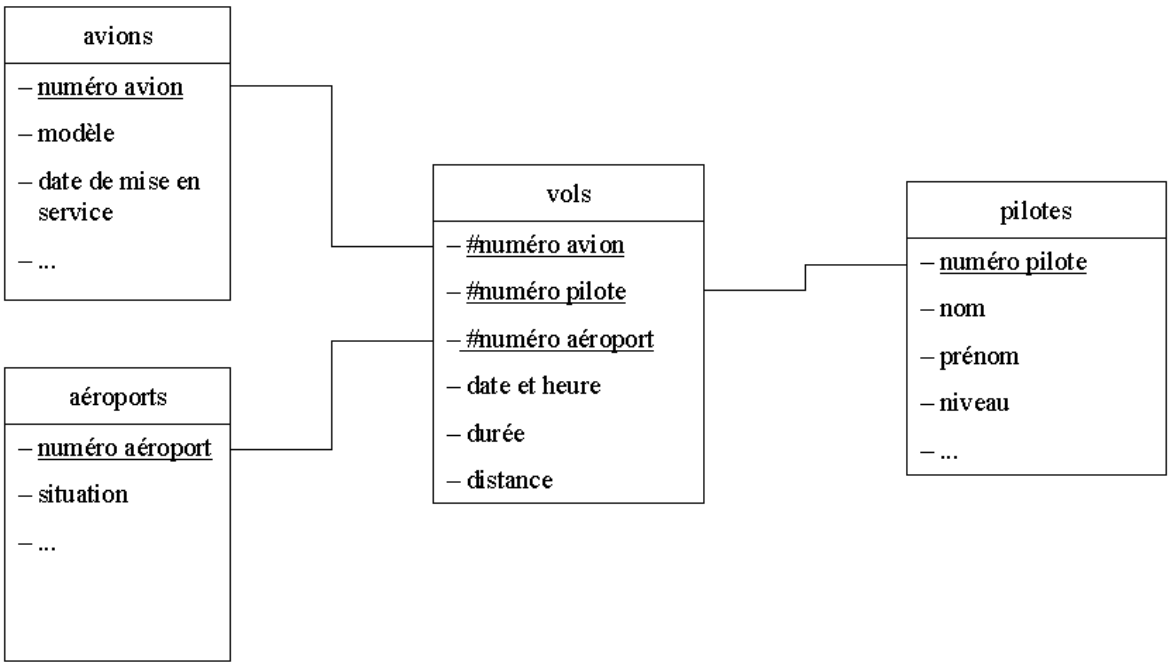

FIG. 15 – *Traduction d'une association ternaire*

3 Modèle physique de données (MPD)

Bien que certains outils (PowerAMC notamment) considère que le MPD et le MLD représentent la même chose, c'est faux. Le MPD est une implémentation particulière du MLD pour un matériel, un environnement et un logiciel donné.

Notamment, le MPD s'intéresse au stockage des données à travers le type et la taille (en octets ou en bits) des attributs du MCD. Cela permet de prévoir la place nécessaire à chaque table dans le cas d'un SGBDR.

Le MPD tient compte des limites matérielles et logicielles afin d'optimiser l'espace consommé et d'optimiser le temps de calcul (qui représentent deux optimisations contradictoires). Dans le cas d'un SGBDR, le MPD définit les index et peut être amené à accepter certaines redondances d'information afin d'accélérer les requêtes.

Par exemple, la table `commandes` de la figure 14 peut être supprimées et ses colonnes, notamment `date`, sont ajoutées à la table `lignes de commandes`. On renonce donc à la troisième forme normale

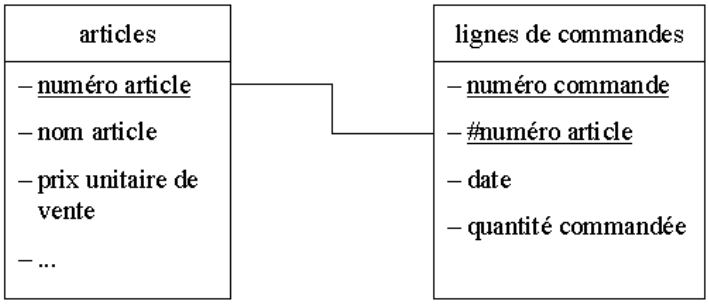

FIG. 16 – *Sacrifice de la troisième forme normale*

puisque la date est répétée autant de fois qu'il y a de lignes dans la commandes, mais on évite une jointure coûteuse en temps de calcul lors des requêtes.

Une application pratique de la séparation entre le MLDR et le MPD est le portage d'une base de données d'un SGBD à un autre. Par exemple, on peut traduire un MPD Access en un MLDR puis traduire ce MLDR en un MPD Oracle.

4 Rétro-conception

Dans la majorité des cas, le travail du concepteur de bases de données consiste non pas à créer une base *ex nihilo*, mais plutôt à corriger ou étendre une base existante. Dans ce cas, la donnée est un modèle physique et la méthode consiste à le traduire en un modèle conceptuel, modifier ce modèle et régénérer le modèle physique modifié. Il s'agit de *rétro-conception* ou *reverse engineering*.

Dans le cadre des bases de données relationnelles, il suffit de traduire le modèle physique en un MLDR, puis d'appliquer les règles de traduction du paragraphe 2.3 dans le sens inverse.

Règle 1 : les tables dont la clé primaire est non composée (et non clé étrangère) deviennent des entités (les colonnes non clés étrangères deviennent des attributs et la clé primaire devient identifiant).

Règle 2 : les tables dont la clé primaire est composée (exclusivement) de clés étrangères deviennent des associations n-aires, où n est le nombre de colonne définissant la clé primaire (les autres colonnes non clés étrangères deviennent attributs).

Règle 3 : les colonnes clés étrangères restantes deviennent des associations binaires de type 1 : n (il reste à ré-inventer leur nom).

Règle 4 : les cardinalités minimales sont 1 si figure la mention `(non vide)` et à retrouver par le bon sens sinon (0 par défaut). Les cardinalités maximales sont 1 ou n selon que la mention `(unique)` figure ou non.

5 Compléments

Les extensions présentées dans cette section font partie de la version 2 de Merise (cf. [4]). Elles permettent de traiter certaines situations réelles plus simplement.

5.1 Agrégation

Exemple : dans un entreprise dont les représentants vendent des produits dans différentes régions, une des règles de gestion est qu'un produit pour une région donnée, ne peut être vendu que par un seul représentant.

Si on se contente du schéma de la figure 17, alors cette règle n'est pas forcément respectée. Il faut

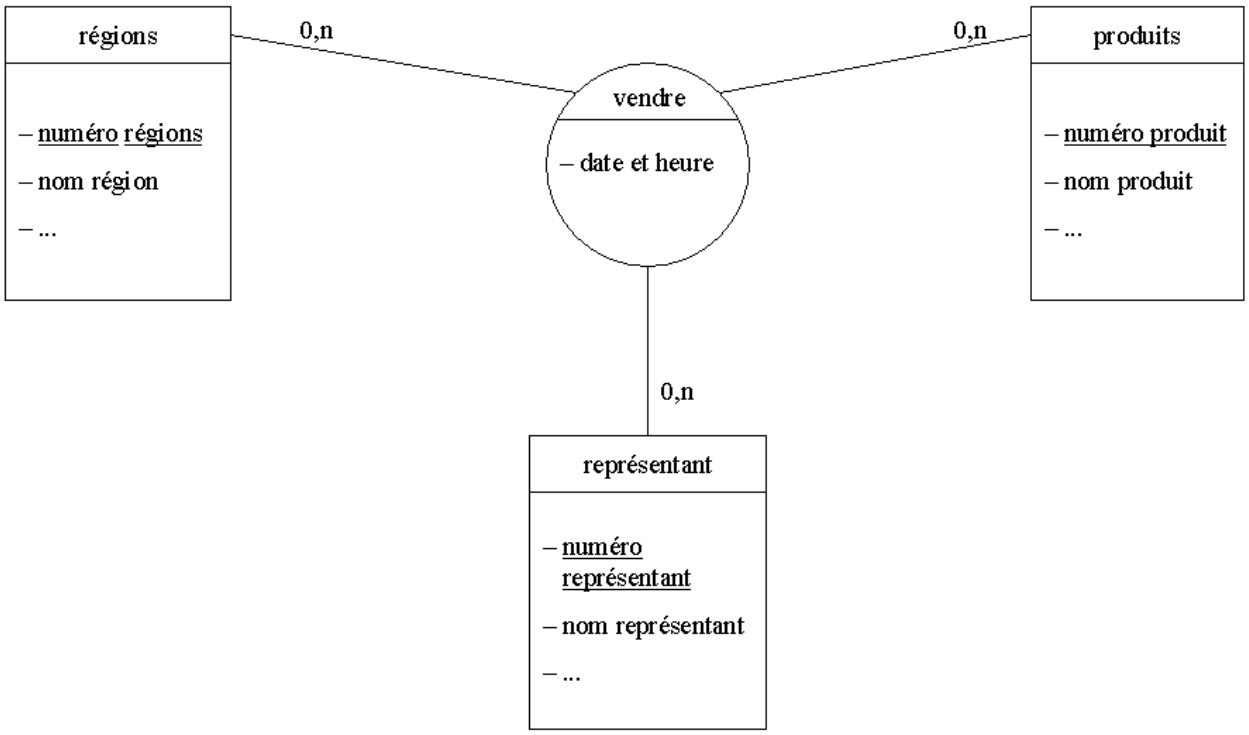

FIG. 17 – *Mauvais MCD*

utiliser une *agrégation* qui permet d'associer une entité à un couple d'entités en associations (cf. figure 18). L'agrégation constitue alors une entité dont l'identifiant est composé des identifiants de ses propres entités en association.

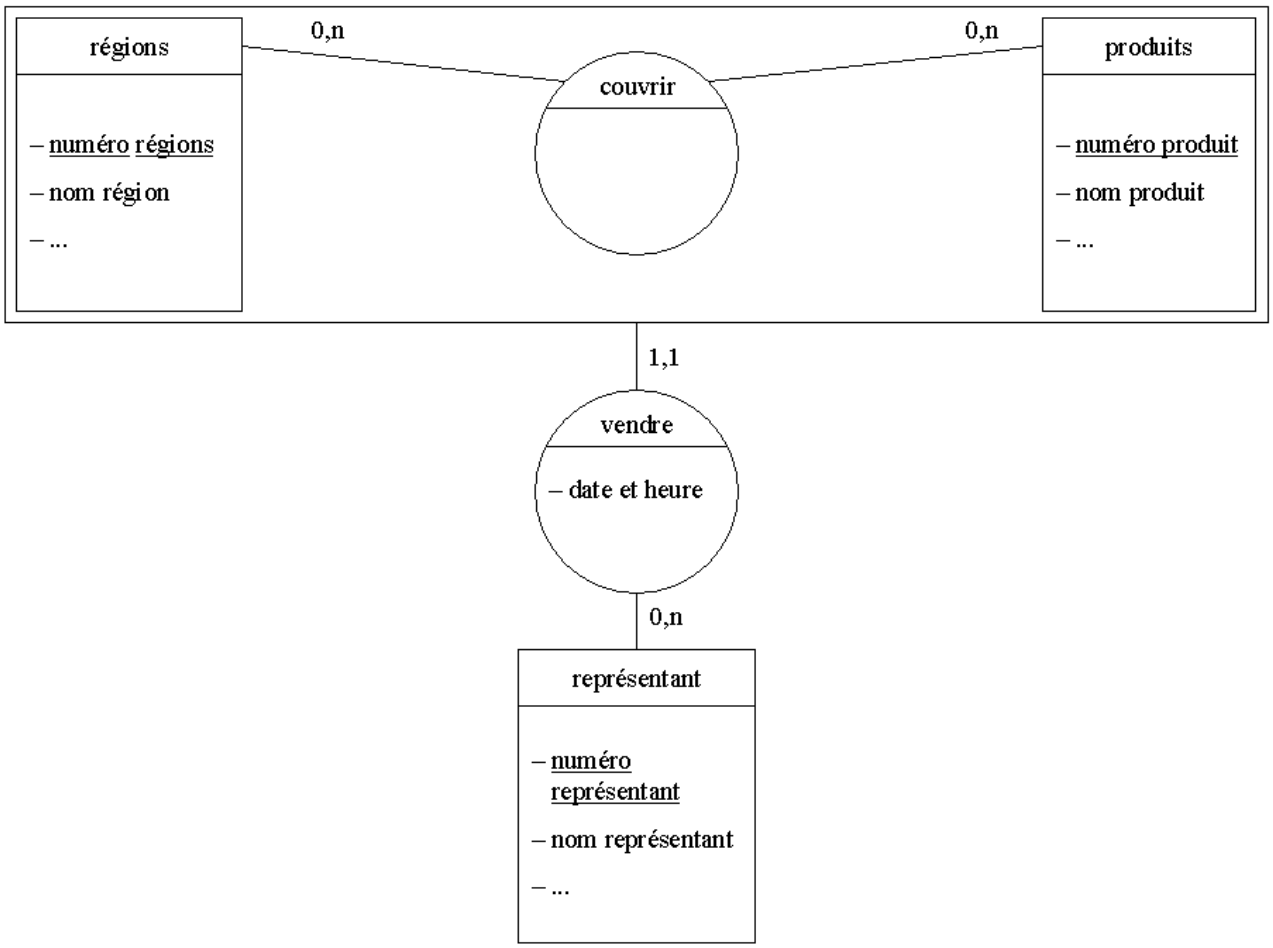

FIG. 18 – *Solution avec agrégation*

Comme l'association `vendre` est de type 1 : n, le schéma relationnel que l'on obtient est alors le suivant :

```
représentants(numéro représentant, nom représentant, ...)
régions(numéro région, nom région, ...)
produits(numéro produit, nom produit, ...)
couvrir(#numéro région, #numéro produit, #numéro représentant (non vide), date et heure)
```

Si l'association vendre était de type n : m (cf. figure 19), alors le schéma relationnel ferait intervenir une table supplémentaire :

```
représentants(numéro représentant, nom représentant, ...)
régions(numéro région, nom région, ...)
produits(numéro produit, nom produit, ...)
couvrir(#numéro région, #numéro produit)
vendre(##numéro région, ##numéro produit, #numéro représentant, date et heure)
```

5.2 Identifiant relatif

Exemple : une entreprise du bâtiment numérote les factures relatives à un chantier par le numéro du chantier suivi d'un numéro automatique. Les factures du chantier 14 sont 1401, 1402 et 1403 tandis que celles du chantier 15 sont 1501 et 1502.

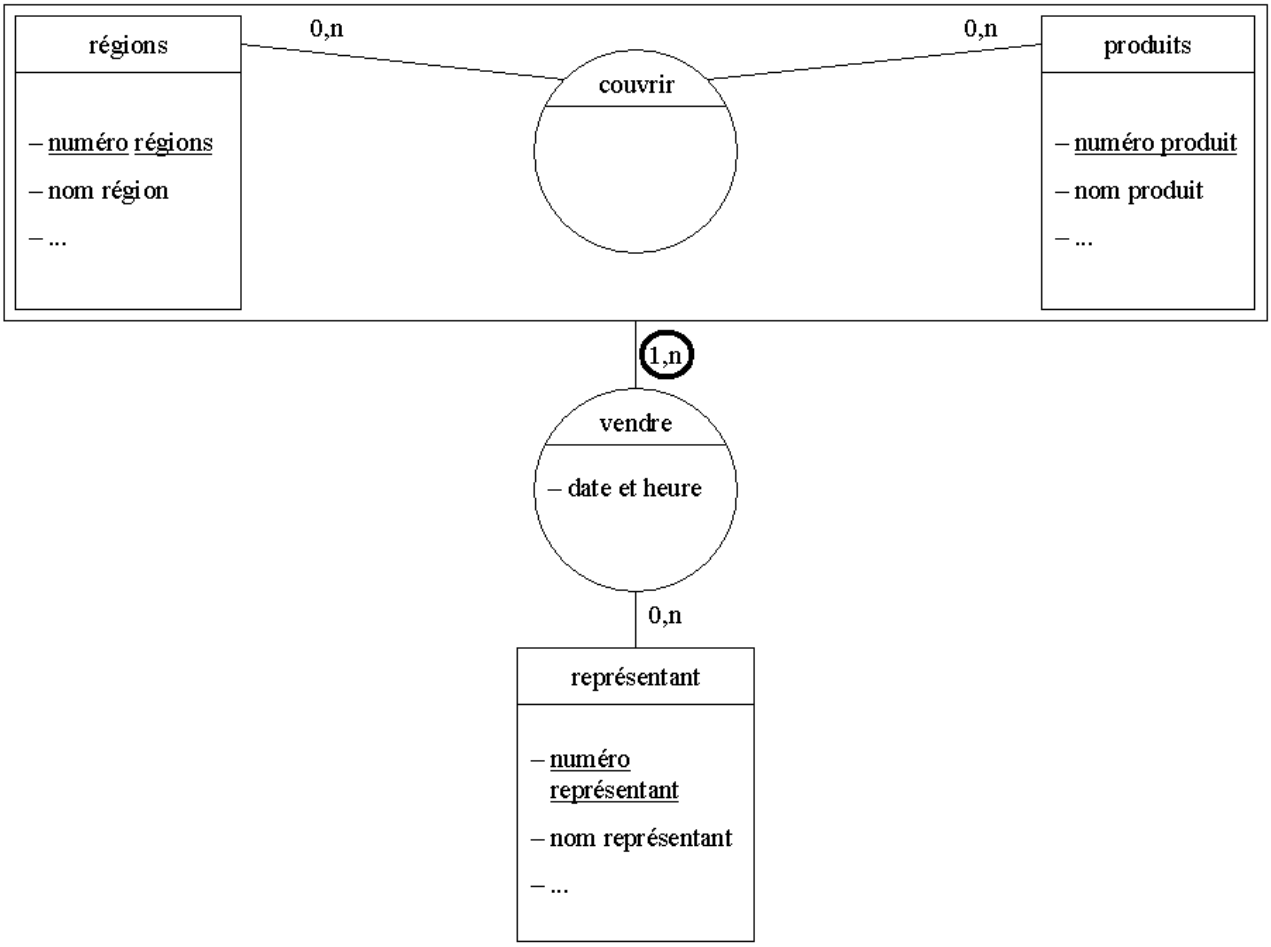

FIG. 19 – *Agrégation avec une association de type n : m*

Le numéro de facture est donc *relatif* au numéro de chantier. On aimerait avoir le schéma de la figure

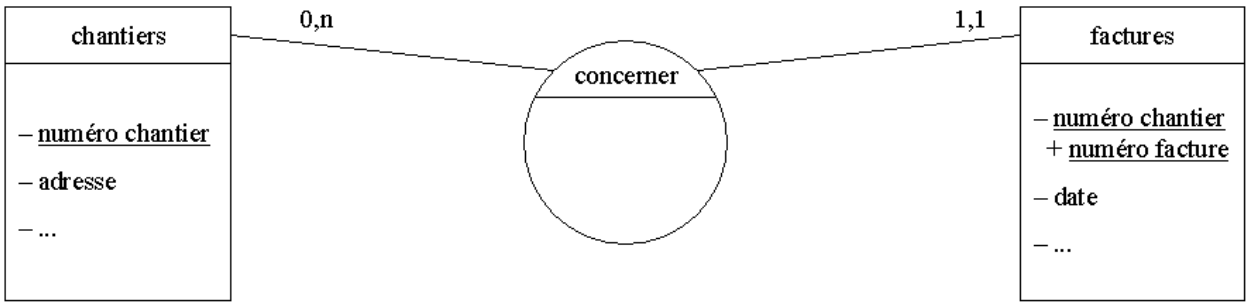

FIG. 20 – *Identifiant par concaténation*

20, mais on ne peut pas utiliser deux fois un même attribut dans un MCD.

Avec Merise 2, il suffit de mettre entre parenthèses la cardinalité 1,1 (cf. figure 21) pour indiquer que l'identifiant de l'entité concernée est relatif à l'autre entité en association.

Au niveau logique relationnel, cela se traduit par une clé primaire composée notamment d'une clé étrangère :

```
chantiers(numéro chantier, adresse, ...)
factures(#numéro chantier, numéro facture, date, ...)
```

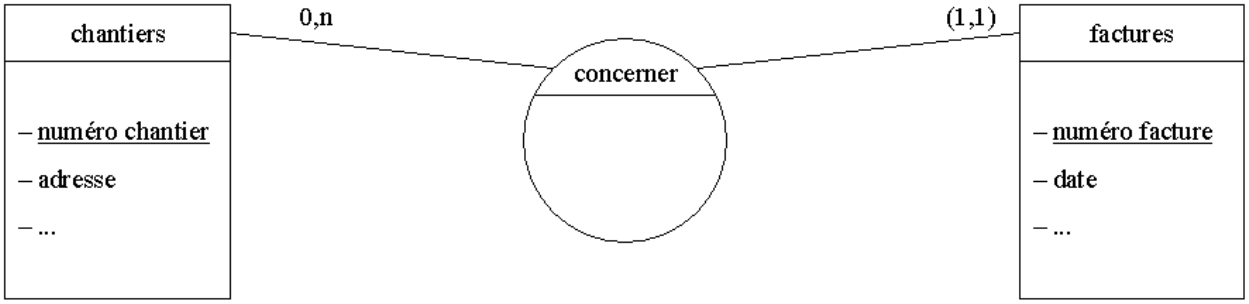

FIG. 21 – *Représentation d'un identifiant relatif*

5.3 Sous-entité

Exemple : les factures d'une entreprise font l'objet d'un règlement par chèque ou par carte. Cette entreprise souhaite connaître pour tous les règlements, leur date, pour les règlements par chèque, le nom de la banque et le numéro du chèque et pour les règlements par carte, le numéro de la carte, le nom de la banque et la date d'expiration.

On a donc une entité *générique* `règlements` et deux entités *spécialisées* `chèques` et `cartes`. Ces deux *sous-entités* de l'entité `règlements` ont des attributs propres mais pas d'identifiants propres.

Sur le schéma entité-association, on représente le lien qui unie une sous-entité à son entité générique par une flèche (cf. figure 22).

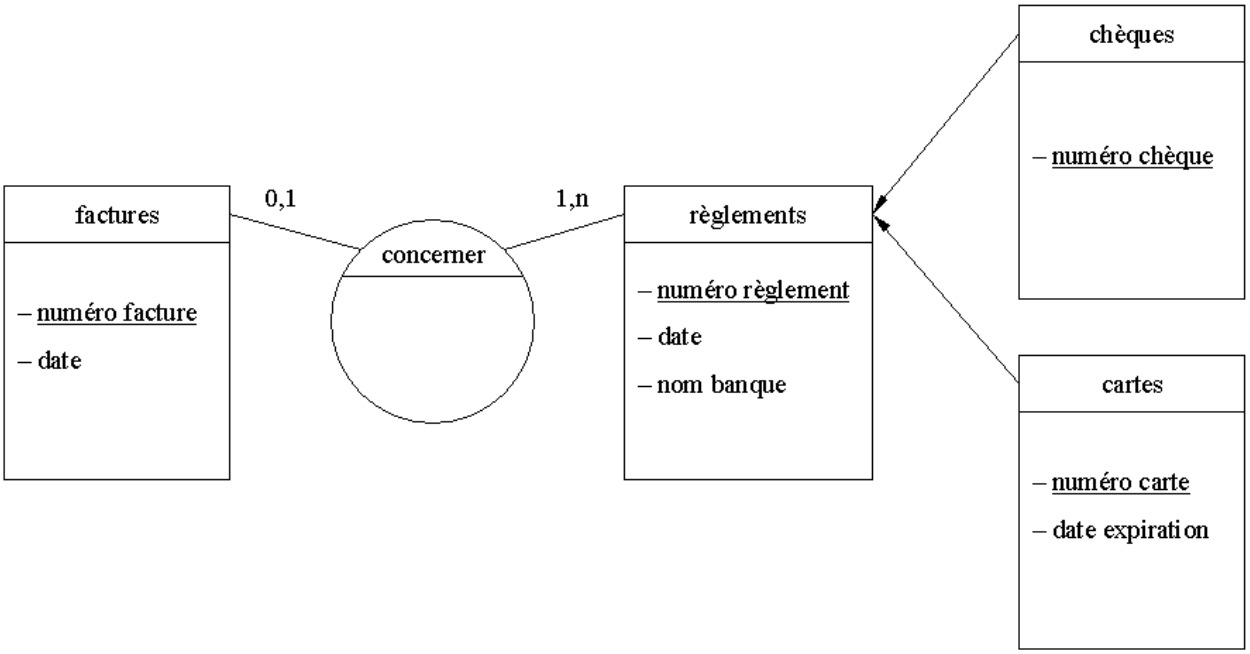

FIG. 22 – *Représentation des sous-entités*

La traduction des sous-entités au niveau logique relationnel fait intervenir une clé primaire commune qui est aussi étrangère :

```
règlements(numéro règlement, date, nom banque)
chèques(#numéro règlement, numéro chèque)
cartes(#numéro règlement, numéro carte, date expiration)
```

Remarque : au niveau logique objet, on retrouve la notion d'héritage.

5.4 Sous-association

Exemple : une entreprise artisanale vend non seulement des produits à prix unitaire fixe, mais aussi des produits sur mesure dont le prix unitaire est calculé à partir de la durée de confection et d'un taux horaire.

Dans ce cas, non seulement l'entité `produits` est spécialisée en `produits standards` et `produits personnalisés`, mais en plus, l'association `concerner` entre les entités `commandes` et `produits` est spécialisée selon qu'il s'agit d'un produit standard ou personnalisé (cf. figure 23).

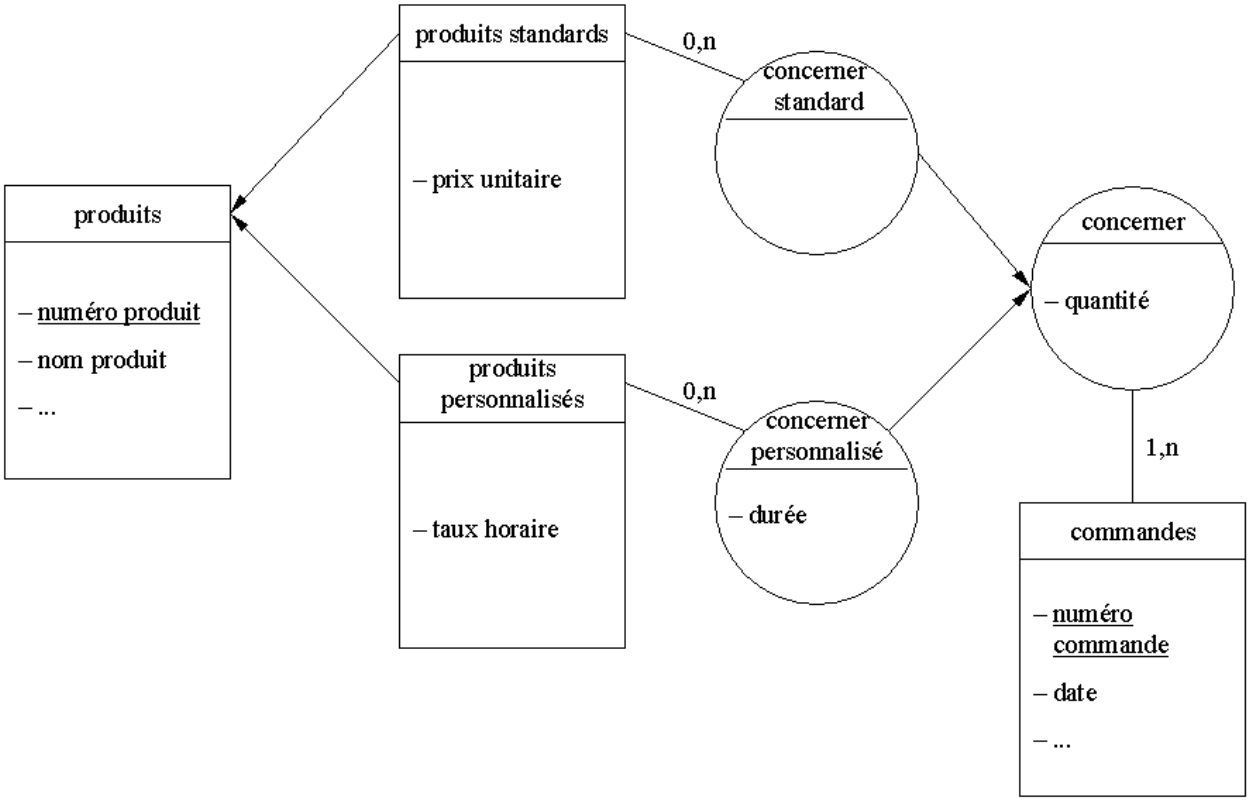

FIG. 23 – *Représentation des sous-associations*

On a alors les *sous-associations* `concerner standard` et `concerner personnalisé` dont le lien avec l'association générique `concerner` est représenté par une flèche.

Dans le schéma relationnel, les sous-associations sont traduites de la même manière que l'association générique correspondante, mais avec leurs attributs propres :

```
produits(numéro produit, nom produit, ...)
produits standards(#(1)numéro produit, prix unitaire)
produits personnalisés(#(2)numéro produit, taux horaire)
commandes(numéro commande, date, ...)
concerner(#numéro commande, #numéro produit, quantité)
concerner standard(##numéro commande, ##(1)numéro produit)
concerner personnalisé(##numéro commande, ##(2)numéro produit, durée)
```

Conclusion

Intérêts de la décomposition MCD/MLD/MPD :
- le MCD permet d'éviter certaines erreurs de conception (en contradiction avec les règles de normalisation) ;
- le MCD permet de voir facilement quelles associations de type n : m, non binaires ou réflexives sont en présence (c'est important) ;
- le MLD peut être obtenu automatiquement par des outils de génie logiciel ;
- le MCD peut être traduit en différents MLD cohérents (notamment on peut traduire un MCD en un MLDR puis en une base de données Access tandis qu'en parallèle le MCD est traduit en un ensemble de classes Java (MPD orienté objet) afin de développer une application web sur cette base Access).

Cependant, la méthodologie Merise est typiquement française. En Grande-Bretagne, la méthodologie standard s'appelle SSADM (Structured Systems Analysis ans Design Method) et repose sur les mêmes principes. Les nord-américains utilisent ce qu'on appelle des diagrammes de flux dont les principes sont repris par la version 2 de Merise.

Aujourd'hui, ce sont les méthodologies objets et leur unification UML (Unified Modeling Language, autrement dit langage unifié de modélisation objet) qui tendent à remplacer Merise et ses extensions objets. Le lecteur est donc invité à se documenter sur le cadre UML pour être à la pointe de l'état de l'art en méthodologie de conception.

Table des figures

Références

[1] Gabay, J. *Apprendre et pratiquer Merise*. Masson, 1989.
Ce livre très synthétique permet de s'exercer sur la méthode.

[2] Matheron, J.-P. *Comprendre Merise*. Eyrolles, 1994.
Cet ouvrage très accessible permet vraiment de comprendre la méthode.

[3] Nanci, D., Espinasse, B., Cohen, B. et Heckenroth, H. *Ingénierie des systèmes d'information avec Merise*. Sybex, 1992.
Cet ouvrage complet détaille la méthode dans son ensemble.

[4] Panet, G., Letouche, R. et Tardieu, H. *Merise/2 : Modèles et techniques Merise avancés*. Édition d'organisation, 1994.
Ce livre décrit la version 2 de la méthode.

[5] Tardieu, H., Rochfeld, A. et Coletti, R. *La méthode Merise. Principes et outils*. Édition d'organisation, 1986.
Il s'agit-là du livre de référence par les auteurs de la méthode.

Index